Trabajo de Fin de Grado

Identidades Parciales
Partial Identities

Alumno: Pablo Martín Huertas

Tutor: Juan Carlos Fabero Jiménez

Doble Grado en Ingeniería Informática y Matemáticas

26 de junio de 2020

Prólogo

"La pieza fundamental de cualquier sociedad es el individuo."

El concepto de sociedad es intrínseco a la naturaleza del ser humano. Debido a que todos formamos parte de algún grupo social, es esencial dar respuesta a las preguntas de quiénes somos y qué nos diferencia de los demás. Estas cuestiones dieron lugar a la génesis del concepto de Identidad. Las personas somos el conjunto de características que nos identifican: nuestra edad, sexo, estatura, comportamiento, lugar de residencia, progenitores, afiliación a grupos sociales... Todos estos rasgos representan la esencia de nuestro ser y son los que dictan cuál es nuestro lugar en la sociedad a la que pertenecemos.

La posesión de todo aquello que forma nuestra propia identidad es vital, desde el uso de un pasaporte o documento nacional de identidad a la hora de viajar hasta el uso de un carné de conducir para poder circular en un vehículo, las personas estamos constantemente empleando los credenciales que tenemos. Hoy en día más que nunca tenemos la necesidad de administrar nuestros credenciales para realizar todo tipo de trámites dentro de la sociedad en la que vivimos. He aquí el objetivo de este trabajo, permitir de manera selectiva y fiable la identificación de credenciales a través de internet.

El resultado buscado es hacer al individuo el único soberano de su propia identidad y permitirle la tarea de identificación. Existen una infinidad de aplicaciones a raíz de la identificación de credenciales en línea, como ejemplo, basta imaginar la posibilidad de realizar todo tipo de trámites gestionados por la administración pública de un país desde la comodidad de una casa.

Para lograr este objetivo, en este trabajo proponemos la remodelación del concepto de red de confianza. Dichas redes de confianza son un sistema ideado por *Phil Zimmermann* en 1992. Su finalidad era resolver el problema de la asociación clave-usuario que planteaba la criptografía de la época. La idea que vamos a plantear es la actualización de estas redes mediante el uso de la novedosa tecnología *blockchain* para así poder resolver gran parte de los problemas que surgían en el sistema tradicional.

Palabras clave

Identidad, verificación de credenciales, criptografía, infraestructura de clave pública, red de confianza, autoridad de certificación, blockchain, certificado X.509, DID, transacción digital.

Abstract

"The cornerstone of any society is the individual."

The concept of society is intrinsic to the nature of the human being. Because we are all part of some social group, it is essential to answer the questions of who we are and what sets us apart from others. These questions gave rise to the genesis of the concept of Identity.People are the set of characteristics that identify them: their age, sex, height, behavior, place of residence, parents, affiliation to social groups ... All these traits represent the essence of our being and are what dictate what is our place in the society to which we belong.

The possession of everything that forms our own identity is vital, from the use of a passport or national identity document when traveling to the use of a driving license to circulate in a vehicle, people are constantly using the credentials that they own. Today more than ever we have the need to manage our credentials to carry out all kinds of procedures within the society in which we live. Here is the objective of this work, to selectively allow the identification of credentials through the internet.

The desired result is to make the individual the sole sovereign of their own identity and allow them the task of identification. There are an infinity of applications that make use of the identification of credentials online, as an example, just imagine the possibility of carrying out all kinds of procedures managed by the public administration of a country from the comfort of a home.

To achieve this goal, in this work we propose the remodeling of the concept of Web Of Trust. Such Webs Of Trust are systems devised by Phil Zimmermann in 1992. Their purpose was to solve the key-user association problem posed by the cryptography of the time. The idea that we are going to propose is updating these webs by using the new technology *blockchain* in order to be able to solve many of the problems that arose in the traditional system.

Keywords

Identity, credential verification, cryptography, public key infrastructure, web of trust, certificate authority, blockchain, X.509 certificate, DID, digital transaction.

Contenidos

Capítulo 1

Introducción

1.1. Motivación del trabajo

Actualmente, la seguridad en el paso de información por medios electrónicos es esencial para nuestra sociedad. Desde las transacciones de un banco hasta la base de datos de un centro médico, la protección y autenticidad de la información supone una necesidad vital para su correcto funcionamiento. Originalmente, internet no fue diseñado para ser una herramienta de comunicación segura por lo que a lo largo de la historia se han tenido que ingeniar diversas técnicas que garanticen dicha seguridad en las comunicaciones. Podemos extraer las siguientes cuatro características fundamentales:

- **Autenticación:** poder verificar que el usuario que está enviando el mensaje realmente es quien dice ser.
- **Confidencialidad:** asegurar que la información enviada tan solo puede ser leída por el remitente y el destinatario.
- **Integridad:** imposibilitar la alteración de la información enviada por un usuario ajeno a la comunicación.
- **No repudio:** no permitir que el remitente pueda negar haber mandado la información.

Este problema se termina por resolver mediante los algoritmos de **clave pública y clave privada**[1] desarrollados entre los años 1970-1980. Dichos algoritmos son capaces de asegurarnos las características de confidencialidad, integridad y no repudio; sin embargo, esto plantea un serio reto.

¿Cómo podemos asegurar que una clave pública realmente pertenece a un individuo en concreto?

Tengamos en cuenta que una vez que seamos capaces de asegurar dicha afirmación ya podremos comunicarnos con ese individuo con garantía de que se cumplen las cuatro características que buscábamos.

El problema se resolvería de forma trivial si conociésemos al individuo en persona, pues bastaría intercambiar las claves públicas en mano y así no quedaría ninguna duda de que dicha clave le pertenece. Ahora el problema es claro: ¿qué ocurre si no podemos hacer dicho intercambio físicamente? ¿Cómo podríamos asegurar quién es el propietario de dicha clave?

Para abordar este problema la comunidad científica y las grandes empresas tecnológicas de la época desarrollaron el sistema **PKI**[2]. Básicamente se traba de un estándar tanto hardware como software que

[1] Principalmente con la invención del algoritmo RSA (*Rivest–Shamir–Adleman*) cuyo poder de encriptación se basa en el concepto de *One-way trapdoor function*, es decir, una función tal que su imagen directa se calcula con complejidad polinomial pero la imagen inversa requiere de un algoritmo con complejidad no polinomial.

[2] Las siglas PKI corresponden a "Public Key Infraestructure", concepto que se explica con más profundidad en la sección 4.1

se fundamenta en la existencia de una organización **centralizada** cuyo objetivo es la administración y la certificación de la correspondencia usuario-clave pública. Esta solución está lejos de la perfección pues plantea una serie de problemas adicionales:

- Toda la confianza reside en la gestión de dicha organización centralizada. Esto plantea una seria dificultad pues aunque la organización tenga una reputación a nivel mundial, seguimos dependiendo de un tercero.
- La gestión que realiza la organización supone un gran coste pues se les atribuye la responsabilidad de conservar y certificar un gran número de relaciones usuario-clave pública. La consecuencia de este hecho es el coste económico que conlleva para el individuo que pide la gestión de la organización.
- La información sobre todas las relaciones usuario-clave pública puede tener asociada información sensible, como por ejemplo el nombre real de una persona o el código de algún documento personal. Esto supone que una vulnerabilidad en la base de datos de la organización plantea un riesgo de filtrado de todos los datos debido a la naturaleza centralizada.

Ante esta serie de problemas, la comunidad científica de la época buscó alternativas a este modelo centralizado. Es aquí cuando aparece la figura de *Phil Zimmermann*, el creador del programa de encriptado PGP[3]. Su gran idea fue el desarrollo del concepto *Web of Trust* o como nos referiremos a él a lo largo de este trabajo, "Red de confianza". La innovación se basa en que ahora disponemos de un modelo descentralizado pues en una red de confianza cualquier individuo es capaz de "dar confianza", es decir, certificar a otro mediante la firma de su clave pública con la clave propia. De esta manera si confiamos en la clave pública de un determinado usuario debido a que hemos comprobado que es quien dice ser, entonces si dicho usuario ha usado su clave privada para encriptar la clave pública de otro individuo y lo verificamos, tendríamos una cierta garantía sobre la confianza de esta última clave. En realidad, más que confiar en la clave del individuo que firma confiamos en su comportamiento, es decir, en su seriedad a la hora de firmar. Confiamos en que no ha firmado una clave de un tercero si no ha verificado previamente su autenticidad.

La idea en sí parece realmente atractiva pero ésta también conlleva unos problemas muy serios:

- Se trata de usuarios dándose confianza entre sí. Esto supone que no tenemos ninguna organización de gran reputación por detrás que nos garantice que dicha clave pública realmente pertenezca a quien dice ser. Esto supone que para una gran empresa, este método de confianza no es efectivo pues no se pueden arriesgar a confiar en usuarios individuales sin importar cuánta gente haya dado confianza a dicha clave.
- El hecho de dar confianza se "diluye" conforme la cadena se hace más grande. Lo que esto quiere decir es que si tenemos una cadena en la que los siguientes usuarios se dan confianza en orden:

  $A \to B \to C \to D \to E$

  Entonces con cierta seguridad podemos afirmar que si confiamos en la clave pública de A entonces en la medida en la que confiemos en la integridad del individuo que posee dicha clave, confiaremos en una cierta clave pública de B que ha sido firmada por A. Sin embargo, a pesar de que confiemos en el poseedor de la clave pública de A es difícil poder confiar en la validez de la clave pública de E.
- Finalmente es pertinente preguntarse: ¿dónde está el historial con todas las transacciones de "dar confianza"? De hecho esto puede suponer un problema en el sentido de que dicho historial puede estar centralizado en alguna organización en concreto y de nuevo esto supondría que una vulnerabilidad en dicho sistema puede suponer que ciertos datos sensibles de usuarios acaben comprometidos.

Con todo esto podemos concluir que los intentos a soluciones del problema de autenticación (**PKI** y **Redes de confianza**) ambos poseen diversos problemas serios.

[3]PGP corresponde a "Pretty Good Privacy"; un resumen más exhaustivo del concepto se expone en la sección1.2.1

He aquí la razón por la que surge este trabajo. La idea consistirá en desarrollar un **modelo descentralizado** que mejore las prestaciones del ya anticuado modelo de las redes de confianza desarrollados por *Phil Zimmermann*. La clave para conseguirlo va a ser replantear el modelo mediante el uso de la innovadora tecnología **blockchain** pues ésta nos va permitir resolver algunos de los problemas que tienen las redes de confianza tradicionales. Con todo esto conseguiremos nuestro objetivo final, la verificación de credenciales *online* **(Identidades parciales)**, gracias a que habremos conseguido asegurar las cuatro características que constituyen una comunicación segura.

La inspiración para este trabajo surge del proyecto "Rebooting the Web of Trust"[4], el cual comenzó en 2015 y sigue en pleno desarrollo en la actualidad. Su creación comenzó con el *white paper* "Rebranding the Web of Trust"[5] y su objetivo es remodelar las redes de confianza para solventar parte de sus problemas gracias a las prestaciones que nos proporciona la tecnología *blockchain*.

[4]Un resumen de la situación actual de dicho problema se encuentra en la sección 1.2.2

[5]Dicho artículo fue escrito por Shannon Appeldine, Dave Orocker, Randall Farmer y Justin Newton y se puede encontrar una referencia a él en el apartado [1] de la bibliografía.

1.2. Estado del arte

Para poder entender los objetivos que vamos a proponer para este trabajo, es esencial hacer un estudio exhaustivo de cómo se encuentra la investigación respecto al tema de las identidades parciales mediante *Blockchain*. Nuestro propósito es avanzar en el desarrollo de esta nueva tecnología, haciendo una aportación a la misma.

El estudio se ha dividido en tres partes:

- **Redes de confianza tradicionales:** comenzamos por ver cómo se definió la *Web of Trust* original para poder así entender en qué áreas pretendemos mejorar.
- **Rebooting the Web of Trust:** analizamos el proyecto que comenzó la idea que subyace a este trabajo, pues en él se propone la remodelación de las redes de confianza para mejorar sus prestaciones.
- **Prototipos:** actualmente existe un importante impulso[6] en el desarrollo de proyectos basados en algún tipo de red de confianza mediante el uso de *blockchain*, por lo que será interesante estudiarlos. Hay que tener en cuenta que no existen implementaciones definitivas en este momento pues se trata de una tecnología que todavía está en desarrollo.

1.2.1. Web of Trust - PGP 2.0

El desarrollo comienza con la publicación del programa PGP *(Pretty Good Privacy)* por parte de *Phil Zimmerman* en 1991. Se trata de un algoritmo híbrido (hace uso de algoritmos de cifrado de clave simétrica y asimétrica) que permite la encriptación, desencriptación y firma de textos, correos electrónicos, ficheros, directorios y particiones. Su esquema de funcionamiento se resume en la siguiente figura:

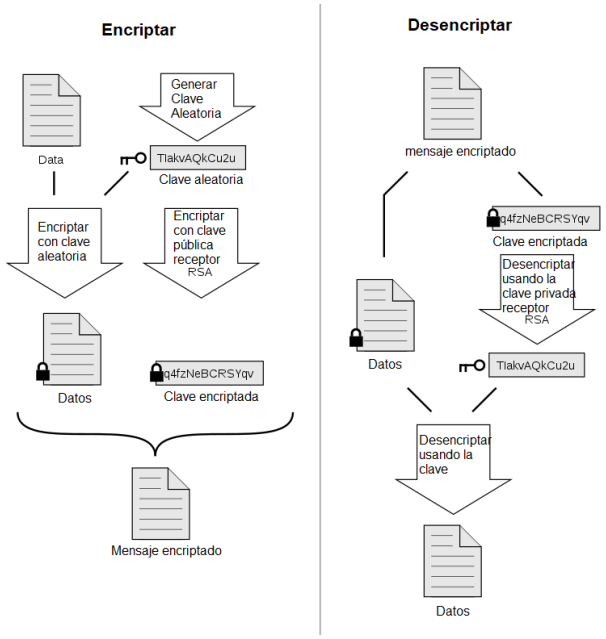


Figura 1.1: Esquemas de los procesos de encriptación y desencriptación de PGP.

Vemos que por un lado los datos se encriptan mediante la clave de criptografía simétrica que tiene una menor complejidad computacional que la asimétrica, mientras que esta última se utiliza para encriptar únicamente la clave generada (o clave de sesión).

[6] De entre los proyectos que estudiaremos algunos están dirigidos por empresas de gran relevancia como IBM o Microsoft. Además cabe añadir que muchos de los prototipos se encuentran en versiones alfa/beta cerradas debido a la competencia sobre este innovador concepto.

El principal problema surge rápidamente: **¿cómo podemos asegurar que una clave pública realmente pertenece a un individuo en concreto?**

El intento de solución al problema es llevado a cabo por *Phil Zimmerman* de nuevo al año siguiente 1992. Se trata de la creación de las redes de confianza *(Web of Trust)*, un sistema de reputación que permita a usuarios "dar confianza" a las relaciones usuario-clave pública. Este sistema contrasta con los sistemas basados en certificados X.509, los cuales están pensados para ser controlados por una CA *(Certificate Authority)*. Aquí tenemos un ejemplo de centralización (CA que mantiene las conexiones usuario-clave pública) frente a descentralización (Red de confianza donde los usuarios dan reputación a las relaciones usuario-clave).

En las redes de confianza se puede apreciar que las relaciones son del tipo "peer-to-peer".

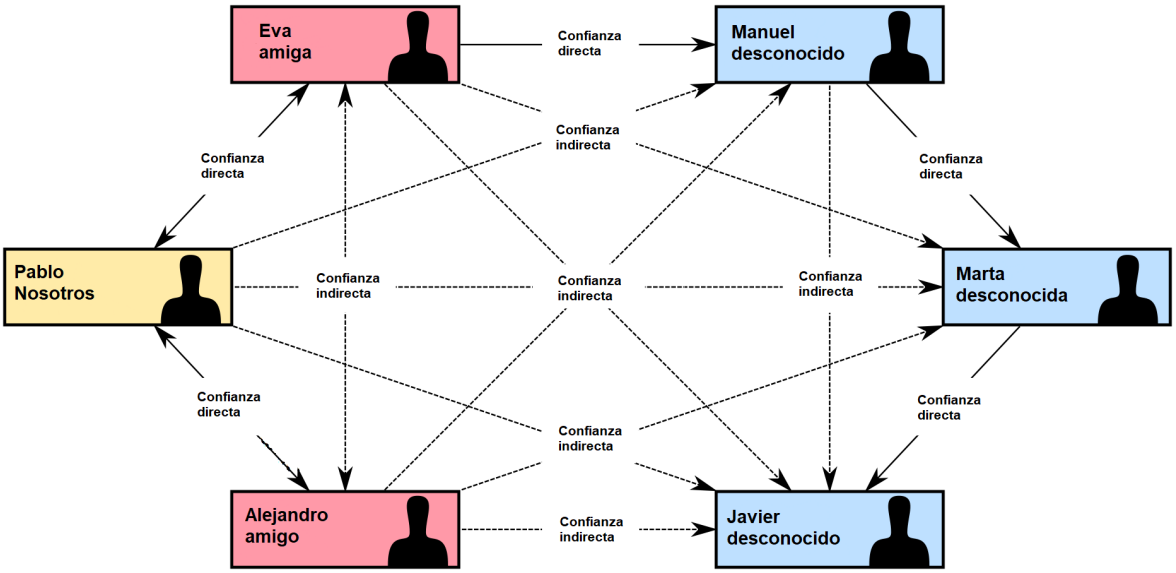


Figura 1.2: Ejemplo de conexiones entre usuarios de una red de confianza.

1.2.2. Rebooting Web of Trust

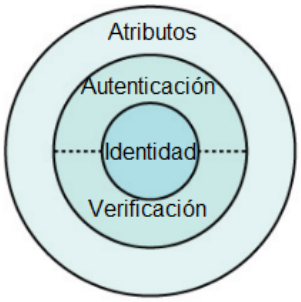
El proyecto *Rebooting Web of Trust* comenzó su desarrollo de manera oficial el día 23 de diciembre de 2015 con la publicación de dos *white papers*[7]: *Rebranding the Web of Trust* y *Creating the New World of Trust*.

En el primero de ellos, *Rebranding the Web of Trust*, se indica la necesidad de buscar mejoras a la anticuada tecnología de las tradicionales redes de confianza desarrolladas por *Phil Zimmerman* en PGP 2.0 de 1992. Para ello se propone una modernización en los dos aspectos fundamentales de las redes de confianza.

- **Red:** en este caso nos centramos en las conexiones entre usuarios, las cuales se producen con un sistema "peer-to-peer". He aquí la necesidad de utilizar la tecnología *blockchain* pues será esencial poseer una **descentralización** total, para poder garantizar la seguridad del sistema.
- **Confianza:** este aspecto es el más difícil de definir, en el caso de las redes de confianza de *Phil Zimmerman* existen utilidades de validación de claves/firmas, validación de identidades, cálculo de reputación,... En *Rebooting the Web Of Trust* se pretenden redefinir estos conceptos para adaptarlos al nuevo entorno con *blockchain*.

[7]Referencias a los *white papers* originales se pueden encontrar en la secciones [1] y [2] de la bibliografía

La propuesta para modernizar las redes de confianza es mediante un modelo basado en dos conceptos: **Entidad** y **Acción**.



- **Entidad:** representa a una persona, un lugar o una cosa. Está compuesta de cuatro partes:
  - **Identidad:** viene representada por un *token* que tiene un identificador visible como puede ser un correo electrónico.
  - **Verificación:** es el proceso que verifica que la identidad es correcta.
  - **Autenticación:** es el proceso que comprueba que un individuo tiene control de un identificador concreto.
  - **Atributos:** se trata de las características o propiedades que pertenecen a la identidad, como pueden ser documento nacional de identidad, carné de conducir, edad, sexo, ... Estos atributos se crean y se evalúan mediante métodos mecánicos y permiten al usuario compartir solo aquellas datos que desee.
- **Acción:** son las interacciones que tienen lugar entre las entidades. Éstas vienen definidas en cinco pasos:
  - **Decisión de privacidad:** es la decisión de cada entidad sobre qué **atributos va a mostrar**. En el caso de no elegir ningún atributo se considerará una acción anónima.
  - **Creación de expectativas:** permite a las entidades ver los atributos revelados de la otra parte y decidir qué acción va a realizar.
  - **Experiencia de la actividad:** ejecución de la acción y obtención de resultados.
  - **Interpretación de la actividad:** cada entidad interpreta los resultados obtenidos.
  - **Anuncio de reputación:** tras el análisis de los resultados se decide si eran los esperados.

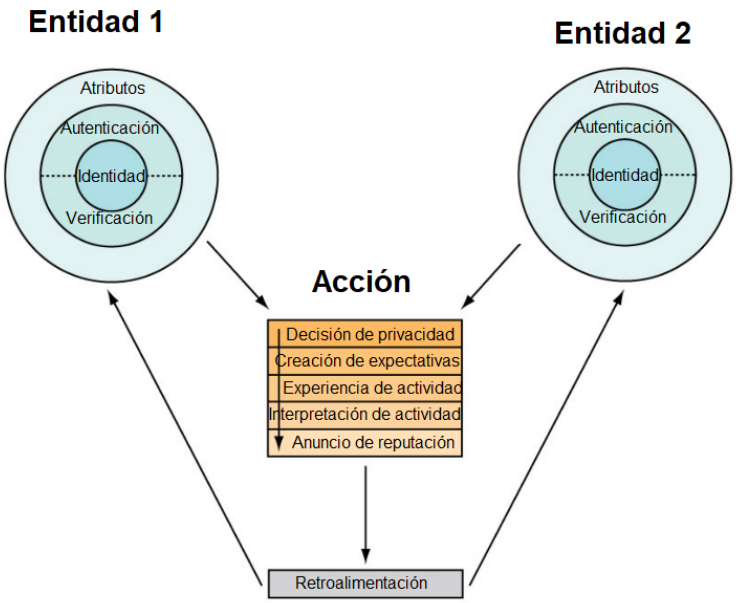


Figura 1.3: Diagrama que representa la interacción entre dos entidades.

1.3. Análisis de la competencia

Un análisis del trabajo que ya está hecho va a ser esencial para entender los objetivos que vamos a fijar en este proyecto. Cabe destacar que debido a novedosa naturaleza de este concepto de anillos de confianza con *Blockchain*, actualmente no existen programas en el mercado; sino que, tan solo hay **prototipos**. Por esta razón, el análisis de la competencia va a resultar dificultoso pues la gran parte de estos prototipos se encuentran en versiones alfa/beta cerradas por razones de competencia. A pesar de todo esto, hay bastante información pública sobre todas estas iniciativas que va a resultar útil.

La estructura de esta sección se divide en dos partes: primero vamos a hacer un estudio individual de los prototipos más influyentes en la actualidad y luego vamos a resumir todos estos datos en una tabla comparativa para poder así extraer conclusiones.

1.3.1. Competencia directa

1.3.1.1. IBM Blockchain Trusted Identity Networks

Este prototipo se centra exactamente en el objetivo de este trabajo, las **identidades parciales**. Se distinguen los siguientes conceptos fundamentales.

- **Individuo:** es el poseedor de la información que le identifica personalmente.
- **Credenciales:** son las características digitales o físicas propias del individuo, ejemplos serían: edad, sexo, dirección de residencia, documento nacional de identidad, carné de conducir, tarjeta de crédito, certificado de nacimiento...
- **Interacciones de identidad:** se trata de las interacciones diarias en las que los individuos muestran sus credenciales, por ejemplo a la hora de pagar con tarjeta es necesario mostrar el número de la misma, para coger un vuelo hay que identificarse con el pasaporte...

En términos de criptomonedas, nuestra *wallet* se compondría de este conjunto de credenciales. Así, el objetivo es permitir una forma de verificar *online* los credenciales de una persona. Además se añade el hecho de que **podemos elegir los credenciales que deseamos mostrar**, esto permite entregar únicamente la información necesaria para así proteger nuestros datos sensibles.

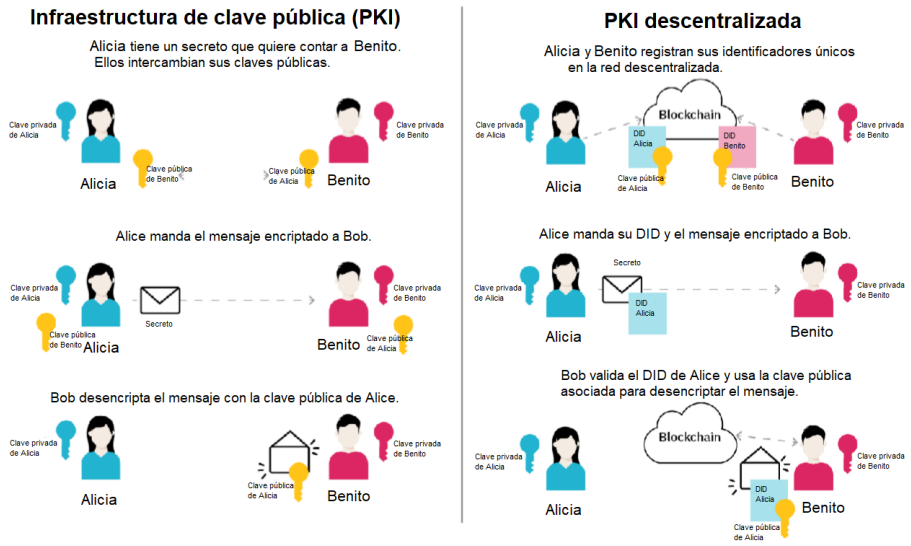


Figura 1.4: Diferencia entre las comunicaciones mediante la PKI tradicional y una descentralizada.

Además, la descentralización aportada por la base en *blockchain* nos aporta dos características esenciales: un registro inmutable con todas interacciones que un individuo ha realizado con su clave pública y un intercambio seguro de claves que el tradicional sistema de PKI no nos permitía.

Una característica esencial del uso de *blockchain* es que la cadena **no** almacena los credenciales de los individuos. La cadena del *blockchain* tiene como objetivo el intercambio de claves públicas, pero los credenciales se han de intercambiar con otro protocolo adicional.

El funcionamiento del sistema de identidades parciales de IBM está basado en que cada individuo es el único propietario de todas las características que le identifican, para ello se definen los siguientes tres componentes.

- **Decentralized Identifier (DID)**[8]: se trata de un tipo de URL que pertenece íntegramente al individuo u organización. Su objetivo es relacionar dicho URL con una identidad para poder asegurar la confianza en la interacción.
- **Credenciales verificables**: son los credenciales que mediante algún estándar deben poder ser verificados.
- **Administración descentralizada de llaves**: cada individuo es poseedor de su clave pública y su clave privada, con ellas se realiza la verificación de las interacciones a través de DID.

La interacción en el *blockchain* se puede resumir en:

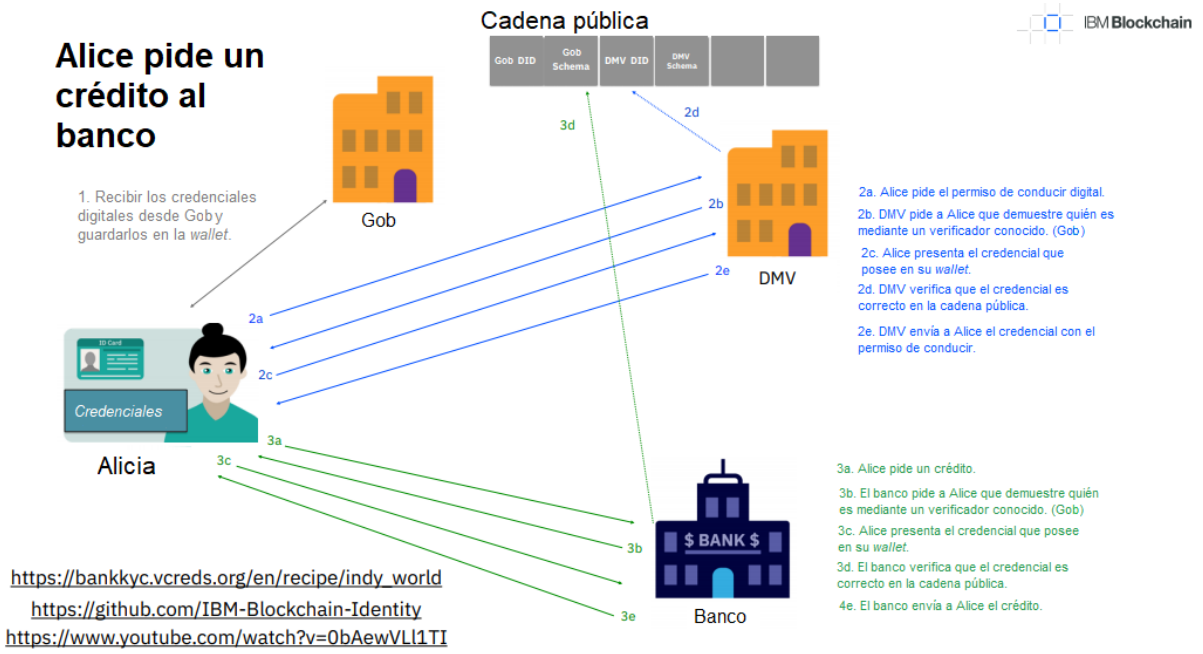


Figura 1.5: Ejemplo de los pasos que se llevarían a cabo para la verificación de credenciales en el sistema.

[8]Los DID serán fundamentales para este trabajo por lo que se hará posteriormente un estudio de los mismos en la sección 2.3.1

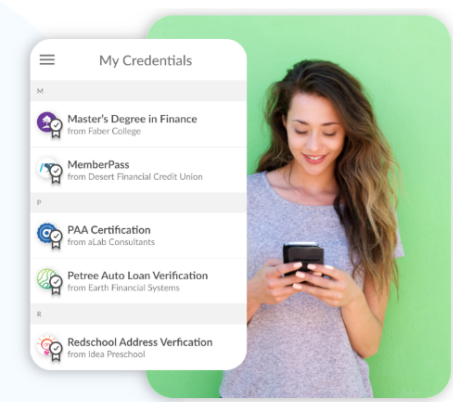
Para llevar a cabo este proyecto, IBM colabora con las siguientes cinco organizaciones.

- **DIF:** desarrolladora del un motor de resolución de DID.
- **W3C:** aporta la especificación que dichos DID deben seguir.
- **Sovrin:** miembro colaborador de la organización DIF.
- **Hyperledger:** proyecto *open source* que proporciona una base de *blockchain*
- **Oasis:** propone el estándar de comunicación de los sistemas de encriptación.

1.3.1.2. Evernym

La compañía Evernym se promociona mediante el eslogan *"The future is built on trust."*. Su principal actividad es proporcionar a clientes con soluciones que permitan la verificación de credenciales digitales. Una de sus características más interesantes es que, al igual que muchos de los otros proyectos que hemos estudiado, Evernym también se basa en código *open Source* y en estándares abiertos (los que indica W3C).

Evernym da mucha importancia al concepto *Self Sovereign Identity*, se trata de algo que ya hemos explicado anteriormente: el individuo viene caracterizado por sus credenciales y el objetivo es que sea el individuo el único propietario de los mismos. De aquí la necesidad nuevamente del uso de DID en algún tipo de sistema descentralizado (sobre todo tipo *blockchain*), para asegurar que el individuo es poseedor de su identidad.



El esquema de funcionamiento de los prototipos Evernym vienen especificados en tres etapas:

- Obtención y verificación de credenciales mediante la herramienta **Verity**.
- Inclusión de dichos credenciales en la *wallet* del usuario para que pueda usarlos mediante la herramienta **Connect.Me**.
- Permitir que esa *wallet* pueda ser empleada por multitud de aplicaciones de terceros mediante **Mobile SDK** que proporciona Evernym.

Una diferencia con el prototipo de IBM es que en caso de Evernym se usa otro tipo de *Open Source* para la base del sistema descentralizado: **Hyperledger INDY**. Dicho proyecto está siendo desarrollado por la *Linux Foundation*.

Además cabe indicar que Evernym colabora con otro de los miembros que participan en el proyecto de IBM, **Sovrin Identity For All**.

1.3.1.3. Veres One

Veres One se define a sí misma como "Una solución usando *blockchain* a medida para optimizar las identidades en la web". De nuevo estamos ante un proyecto de competencia directa pues el objetivo de Veres One es una solución al problema de identidades parciales, en su caso las propiedades que hacen destacar a Veres One del resto de compañías son:

- **Precios bajos:** el *blockchain* de Veres One a diferencia de otros tipo Ethereum que son de propósito general, está centrado únicamente en el problema de identidades parciales, por lo que se ha optimizado la eficiencia en esa tarea.
- **No especulativo:** en otros *blockchain* existe el concepto de *scarce network token* que tiene el efecto de hacer variar de manera difícilmente controlable los precios del sistema, en Veres One este concepto no existe y se asegura que las comisiones sean bajas y estables.
- **Distribuida:** como otras redes descentralizadas, el hecho de que no exista una organización central que controle el sistema tiene una serie de ventajas que ya hemos indicado en varias ocasiones con anterioridad.
- **Permite auditorías:** esto es una consecuencia directa de que la información de la cadena es pública por todos los usuarios de la red.
- **Privacidad:** Veres One en consonancia con la ley de protección de datos europea GDPR, almacena únicamente en la cadena los datos estrictamente necesarios para llevar a cabo la actividad de la aplicación.

Indicamos un aspecto que Veres One aclara en su página web: aunque en muchos de los apartados de este trabajo nos referimos a los credenciales de los "individuos" en realidad también puede tratarse de organizaciones o colectivos.

El funcionamiento de la aplicación de Veres One viene sostenida por dos tipos de participantes:

- **Aceleradores:** pPueden ser gobiernos, organizaciones que proporcionan credenciales o registradores. Su objetivo es colaborar en agilizar los procesos de creación y validación de los credenciales de los individuos. Deben estar de acuerdo con el *"Accelerator Agreement"* desarrollado por la compañía.
- **Nodos:** son esencialmente los verificadores que mantienen la cadena consistente mediante las pruebas de trabajo. Como en otros sistemas *blockchain* existe un convenio de recompensa a los nodos por la generación de los *hash* correctos.

Un aspecto que hace a Veres One especialmente interesante de entre las demás compañías de la competencia es la seriedad con la que tratan el tema de la privacidad. Un ejemplo claro de una medida que tomaron al respecto trata sobre el almacenamiento de los credenciales: el usuario tiene la elección de si decide guardar sus credenciales únicamente en su dispositivo, únicamente en la nube o en ambos.

1.3.2. Competencia parcial

1.3.2.1. IXO - Microsoft

Se trata de un proyecto desarrollado por la fundación IXO. En este caso el objetivo principal no son las identidades parciales (verificación de credenciales en línea), sino que el propósito de IXO es el de

proporcionar un sistema descentralizado *blockchain* que recoja una gran cantidad de datos de calidad y que los analice mediante algoritmos de inteligencia artificial.

La parte polifacética de este proyecto viene en el momento en el que hemos indicado "una gran cantidad de datos", ya que dichos datos pueden ser de cualquier índole. La plataforma IXO nos proporciona una base a partir de la cual cualquier usuario es capaz de crear un proyecto que se adjunta al sistema para recaudar un tipo de datos específicos, por ejemplo puede haber proyectos de: recaudación de datos meteorológicos de una zona, datos sobre el censo de una población...

La razón por la que esta plataforma puede resultar de interés para nuestro trabajo es que implementa una metodología que permite decidir qué miembros participan en un proyecto concreto desplegado en esta plataforma. De esta manera, se podría realizar un proyecto que se ajuste parcialmente a lo que estamos buscando: recaudar credenciales que personas que para participar en este subsistema en concreta tienen que recibir permiso para ingresar (por ejemplo una CA que les verifique)

1.3.2.2. DIF - Decentralized Identity Foundation

La fundación DIF tiene como objetivo principal el desarrollo de los estándares que se han de implementar para la verificación de identidades de sistemas descentralizados. Esto viene en gran medida relacionado con la sección de DID (*Decentralized Identifiers*) que se estudia a continuación en el apartado de desarrollo de este trabajo.

La idea es que históricamente para identificar credenciales es imperativo seguir algún convenio que por ejemplo un gobierno ha dictado, como el formato de un pasaporte. Ahora en el caso de una red descentralizada, necesitamos un formato común a partir del cual implementar nuestro "equivalente al pasaporte".

El trabajo de DIF se compone de:

- **Especificaciones técnicas:** se trata de un convenio que deben seguir los sistemas descentralizados que implementen autenticación de identidades.
- **Implementaciones ejemplares:** lLa fundación también desarrolla prototipos de cómo se han de implementar estas especificaciones para ayudar a la comunidad de programadores.
- **Coordinación de la industria:** DIF trabaja en ayudar a la coordinación de las organizaciones que cooperan en el desarrollo de proyectos sobre la autenticación de identidades en sistemas descentralizados. Por ejemplo, ayudan al proyecto que ya hemos estudiado de *IBM Blockchain Trusted Identity Networks*.

En nuestro caso nos interesa destacar el prototipo de referencia: **DIF Universal Resolver**[9]. Se trata de un programa que, dado un URL del tipo DID, hace una búsqueda en una de las redes descentralizadas existentes. Por ejemplo, podríamos buscar a qué DID-Document apunta una dirección DID en la cadena del Bitcoin con la llamada "`did:btcr:xz35-jznz-q6mr-7q6`", donde `btcr` es el protocolo específico de busqueda en el *blockchain* de Bitcoin.

[9]La herramienta it DIF Universal Resolver tiene una demo accesible online a partir del link: *https://uniresolver.io/*. En ella se pueden ejecutar casos de prueba de búsqueda de DID-documents en diferentes redes descentralizadas.

Para nuestro proyecto vamos a implementar una función parecida pues vamos a tener una cadena en nuestro *blockchain* que contenga los DID-Documents asociados a los DID de los individuos que forman parte de nuestra red. Por ello deberemos programar una función de búsqueda en nuestra cadena análoga a las implementadas por el *DIF Universal Resolver*.

1.3.2.3. ConsenSys

La compañía ConsenSys tiene su negocio basado en consultoría de soluciones tecnológicas basadas en *blockchain*. Su trabajo se divide en los siguientes cuatro sectores:

- **Enterprise:** se trata de la solución de problemas relacionados con *blockchain* que un cliente presente, para ello ConsenSys se especializa en soluciones mediante la plataforma Ethereum[10].
- **Startups:** la innovación es un aspecto esencial para una tecnología tan nueva como las redes descentralizadas, por ello ConsenSys invierte en numerosas startups que desarrollan soluciones *blockchain*.
- **Development:** además de llevar a cabo soluciones en Ethereum, la empresa desarrolla y publica herramientas para dicha plataforma.
- **Education:** finalmente, la compañía ofrece cursos tanto *online* como presenciales para enseñar a trabajar en Ethereum.

En cuanto a la variedad de soluciones ofrecidas es muy amplia: finanzas, energía y sostenibilidad, sector público, comercio internacional... Para nuestro trabajo nos vamos a enfocar en la solución que trata las identidades parciales, **Digital Identity**[3].

Desde las propias publicaciones de la compañía nos indican los tres aspectos fundamentales que tratar en la sección de identidades parciales:[11]

- **Para compañías:** "...Las compañías tienen que recopilar los datos de sus usuarios para poder hacer su trabajo. Esto crea un riesgo frente a las regulaciones de privacidad como pueden ser el GDPR..."
- **Para Internet of Things:** "...Los dispositivos de IoT deben identificar sensores, monitores y otros dispositivos para organizar el acceso a datos sensibles de una manera segura..."
- **Para usuarios:** "...La identidad es central para el funcionamiento de una sociedad. Cada persona está compuesta de una colección de credenciales que le identifican. El objetivo es proporcionar a los usuarios una forma de reclamar la propiedad de su propia identidad para llevar a cabo operaciones *online*..."

Vemos que en las propias palabras de ConsenSys, los problemas que plantea la Digital Identity son realmente interesantes debido a las necesidades que los usuarios tienen en la actualidad. La parte más en línea con nuestro trabajo es la destinada a las usuarios: se trata de permitir la verificación de credenciales *online*. La ventaja de una solución con tecnologías descentralizadas reside en permitir a los usuarios ser los únicos propietarios de su propia identidad y de reducir la cantidad de información personal identificable (nombre, dirección, datos biométricos...) que un usuario tiene que mantener en la red.

[10] Ethereum es una plataforma de *blockchain* que se caracteriza por proporcionar una base a todo tipo de sistemas descentralizados, la clave reside en que en Ethereum lo que se transmite son contratos digitales (que ejecutan chaincode), los cuales los especifica el propio desarrollador, dando así libertad para realizar todo tipo de operaciones.

[11] Todas las citas se han obtenido desde la referencia [3] de la bibliografía.

Tal y como ya hemos vistos en otros casos, de nuevo ConsenSys propone el uso del estándar de DID para sus soluciones relacionadas con identidades parciales. Dichos DID recordamos que son como el "pasaporte" común para que los usuarios de una red descentralizada puedan procesar la información de los demás. Además, una vez que un usuario sepa que cierta clave pertenece a otro individuo, mediante criptografía de clave asimétrica ya tiene garantizado poder comunicarse con él de manera segura.

1.3.3. Conclusiones

Realmente en esta sección hemos estudiado una gran cantidad de información, podemos observar cómo todos los proyectos que hemos mostrado son muy recientes (desarrollados durante los últimos 8 años). Esto nos da un claro indicador de que estamos ante un problema de gran interés y por el que está surgiendo una fuerte competencia para apoderarse del mercado. Es importante ver esto como un claro refuerzo de la predicción de que una tecnología que resuelva el problema de las identidades parciales se va a implantar en nuestra sociedad en un futuro. Ahora bien, además de habernos asegurado que el tema tratado en este trabajo es relevante tanto para la comunidad científica como para la sociedad, nuestro segundo objetivo va a ser extraer las conclusiones clave que nos ayuden a desarrollar un prototipo propio. Para ello y debido a la gran cantidad de información que comprende esta sección vamos a presentar la siguiente tabla con los aspectos más importantes de cada una de las iniciativas.

Proyecto	Trata principalmente el problema de identidades parciales	Hace uso del estándar DID	El proyecto tiene menos de 8 años	Característica distintiva	Emplea tecnología Blockchain (especificar)
IBM	✓	✓	✓	La cadena no posee información personal identificable	✓ Hyperledger Fabric
Evernym	✓	✓	✓	Permite usar su wallet de credenciales en terceros mediante su Mobile SDK	✓ Hyperledger INDY
Veres One	✓	✓	✓	Introduce concepto de Aceleradores para agilizar creación/verificación de credenciales	✓ Hyperledger Fabric
IXO	✗	*	✓	Base multiplataforma que permite desplegar proyectos de todo tipo sobre redes descentralizadas.	✓
DIF	✗	✓	✓	Herramienta auxiliar a proyectos de identidades parciales para resolver la conexión DID/DID-Document	✓ (Múltiples)
ConsenSys	✗	*	✓	Consultora que proporciona variedad de soluciones basadas en blockchain	✓

* *: puede usar el estándar DID dependiendo del proyecto concreto que esté tratando.*

Podemos extraer las siguientes conclusiones para nuestro trabajo:

- Será esencial usar el estándar DID que ha especificado la organización W3C, pues todos los proyectos actuales de gran calibre lo tienen en cuenta.
- Hemos decidido usar el *open source* **Hyperledger Fabric** para tener una eficiente base de nuestro *blockchain*. El funcionamiento de Hyperledger Fabric se estudiará con más detenimiento en la siguiente sección 2 de desarrollo.
- Nos parece de gran interés la filosofía de IBM de no introducir información personal identificable en la cadena pública, por lo que en nuestro prototipo tan solo introduciremos documentos DID. Es decir, la red descentralizada es como un mecanismo auxiliar para resolver el problema de identificar a un individuo con su clave.

Capítulo 2

Desarrollo

2.1. Plan de trabajo

Tras el estudio del estado del arte sobre el tema que abarca este trabajo y habiendo reflexionado sobre las conclusiones que hemos obtenido a partir del análisis de la competencia, ya estamos preparados para el desarrollo de nuestro proyecto. Con el fin de facilitar y mejorar la eficiencia de dicho desarrollo, esta sección constará de los siguientes cuatro bloques fundamentales: propuestas de casos de uso, estudio de las herramientas auxiliares, especificación del sistema y detalles de la implementación.

Más concretamente, cada apartado comprenderá la siguiente información:

- **Casos de uso:** se propondrán una serie de objetivos que un usuario de nuestra aplicación debe ser capaz de realizar. Una vez planteado un objetivo se especificará tanto la secuencia de acciones que el usuario deberá realizar como otras características como pueden ser los requisitos, las precondiciones, el tipo de actor...
- **Herramientas auxiliares:** a la hora de implementar nuestra prueba de concepto propuesta, vamos a hacer uso de herramientas de software libre que se encuentra en la red pues nuestra meta es avanzar en la materia de la autenticación de credenciales en redes descentralizadas a partir de la base que ya nos proporciona la comunidad científica.
- **Especificación:** una vez ya hemos expuesto qué tareas debe ser capaz de realizar el usuario y las herramientas que nuestro sistema va a requerir, podemos pasar a la especificación concreta que indicará el funcionamiento del prototipo que desarrollemos.
- **Implementación:** finalmente, tras la puesta en código de la especificación señalada en el apartado anterior se remarcarán las características principales del mismo para así dar a entender con mayor nitidez lo que hay por detrás del funcionamiento de la prueba de concepto. Además se realizarán pruebas de ejecuciones que confirmen el correcto funcionamiento.

2.2. Casos de uso

El objetivo de esta subsección es aclarar las funcionalidades que se esperará que el prototipo ofrezca. Para ello vamos a exponer una serie de casos de uso reales que aclaren dichas funcionalidades. Este apartado será de gran importancia pues tras él realizaremos un breve esquema-resumen con las acciones realizables que nos permitirá entender las decisiones de diseño de los siguientes apartados.

La estructura de los casos de uso se compone de las siguientes partes:

- **Número del caso de uso y título**
- **Nivel:** este apartado es vital pues la filosofía de cada caso de uso viene dada por el nivel que trata, hay dos posibles:
  - **Objetivo del usuario:** estas acciones suponen aquellas que los usuarios finales deben poder realizar mediante las herramientas que proporcione el prototipo.
  - **Subfunciones del sistema:** estas acciones son "transparentes" al usuario, pues no las ejecuta directamente; sin embargo, serán esenciales para mantener tanto el buen funcionamiento del sistema como asegurar las prestaciones que nuestro prototipo debe ofrecer.
- **Actores:** tanto principales, los que desencadenan la acción, como se secundarios, los que participan en el transcurso de la misma.
- **Objetivo de contexto:** es la meta que el usuario o en su caso la subfunción se proponen a la hora de realizar la acción.
- **Precondiciones:** en el caso de un usuario es el la acción que comienza el caso, mientras que en una subfunción es el desencadenante del caso.
- **Postcondiciones:** son los posibles estados finales tras acabar el caso de uso.
- **Flujo principal:** hace un seguimiento del caso de uso paso a paso.
- **Flujos secundarios:** en caso de haber ramificaciones en los pasos a tomar, se aclararán en este apartado.
- **Frecuencia de ocurrencia:** indicará la frecuencia con la que dicho caso de uso aparecerá en la práctica real.

En las siguientes páginas vamos a presentar la lista con los casos de uso que hemos propuesto como más relevantes para la aplicación.

Caso de uso 1:	**Registro en un canal[1] del blockchain**
Identificador del caso:	USO_REGISTRO_CANAL
Nivel:	Objetivo del usuario
Actor principal:	Usuario final
Actores secundarios:	Miembro del canal
Objetivo en contexto:	Un usuario nuevo a un canal hace una petición de registro para poder tener acceso a las operaciones sobre la cadena distribuida en dicho canal.
Precondiciones:	El usuario realiza una solicitud de registro a un miembro con permisos del canal.
Postcondiciones:	▪ **Éxito:** el miembro receptor de la petición acepta la unión y proporciona al usuario un credencial que se le añadirá a su *wallet* con la identificación. ▪ **Fallo:** es rechazada o no contestada la petición y el usuario recibe un mensaje de fallo al acceder al canal.

Flujo principal:

1. El **usuario** manda una petición de ingreso a un canal.
2. Un **miembro** de dicho canal recibe el mensaje y decide aceptarlo.
3. El **usuario** envía un mensaje al miembro con su clave pública.
4. El **miembro** manda un mensaje de actualización a los demás miembros y verificadores para que otorguen permisos a la clave pública del usuario novel.
5. El **usuario** almacena sus credenciales con la clave pública en su *wallet*.

Flujos secundarios:

2.a El miembro pide información adicional antes de tomar la decisión:

1. El **miembro** manda un mensaje con una petición de datos al usuario.
2. El **usuario** envía los datos pedidos.
3. Se vuelve al paso 2 del flujo principal.

2.b El miembro rechaza la petición.

1. El **miembro** manda un mensaje de rechazo a la solicitud.

Frecuencia de ocurrencia:	Única durante el ingreso a un nuevo canal.

[1]En lo que concierne a esta sección podemos definir de manera intuitiva el concepto de canal como el medio a través del cual un usuario podrá mandar mensajes a los miembros del blockchain. Sin embargo, una definión más formal y precisa se puede encontrar en la sección 2.3.2 que sigue inmediatamente a ésta.

Caso de uso 2:	**Introducción de un DID-document en una cadena**
Identificador del caso:	USO_INTRODUCIR_DIDDOCUMENT
Nivel:	Objetivo del usuario
Actor principal:	Usuario final
Actores secundarios:	Verificadores del canal
Objetivo en contexto:	Un usuario realiza una transacción en la que intenta introducir un DID-document (asociado a la clave pública que tiene en su *wallet*) a la cadena.
Precondiciones:	El usuario realiza una transacción de introducir DID-document.
Postcondiciones:	▪ **Éxito:** los verificadores añaden dicha transacción a un bloque y mediante la prueba de trabajo (generación de *hash*) lo añaden a la cadena. ▪ **Fallo:** la cadena no recibe ningún cambio.

Flujo principal:

1. El **usuario** manda una transacción de introducir DID-document junto con los datos del mismo.
2. Los **verificadores** que estén a la escucha comprueban que el DID-document corresponde a la clave pública que tiene el usuario en su *wallet*.
3. Los **verificadores** añaden la transacción a un nuevo bloque y comienzan el cálculo de la prueba de trabajo (*hash*).
4. Un **verificador** consigue generar el *hash* en cuestión y lo publica para que el resto de verificadores lo comprueben.
5. Los **verificadores**, tras comprobar el nuevo bloque, publican la nueva cadena a todos los usuarios de la red.

Flujos secundarios:

2.a El DID-document corresponde a una clave pública que no pertenece al *wallet* del usuario:

   1. Un **verificador** manda un mensaje de error usuario que realizó la transacción.
   2. El **usuario** recibe un mensaje con la indicación de qué ha fallado.

Frecuencia de ocurrencia:	Única para cada introducción de un DID-document asociado a una clave pública.

2.3. Herramientas auxiliares

2.3.1. DID - *Decentralized Identifier*

Tradicionalmente los sistemas de identificación se basaban en alguna autoridad centralizada como por ejemplo el gobierno de un país, el cual proporciona mediante un estándar que han definido un pasaporte que permite identificarse en todos aquellos lugares donde sea aceptado. Este método de identificación depende esencialmente del convenio que dicha autoridad haya dictado. En un sistema *blockchain* no existe dicha autoridad por lo que es necesario un nuevo estándar de identificación. En las palabras de la organización W3C[2]: "... Un DID es un identificador único y global que no requiere de ninguna autoridad registradora pues su propio registro se hace a través de tecnología *blockchain* o de alguna otra forma de red descentralizada..."

Así el objetivo de los DID es otorgar al individuo de un identificador con una característica especial y es que dicho individuo debe ser capaz de demostrar que él controla el correspondiente DID. La clave reside en que esta demostración se debe poder llevar a cabo sin presencia de ninguna organización centralizada.

Un DID en esencia es una relación entre los dos siguientes conceptos:

- **DID-Subject:** se trata de la entidad que tiene el control del DID, comúnmente es un individuo o una organización.
- **DID-Document:** es un conjunto de datos que describen al DID-Subject, incluye mecanismos como pueden ser claves públicas que permiten al DID-Subject autenticarse y demostrar que está en control del DID en cuestión. Adicionalmente un DID-Document puede poseer otras características no especificadas del DID-Subject.

Por otro lado tenemos el concepto de *DID methods*, se trata de un conjunto de mecanismos que permiten manipular (crear, leer, actualizar y desactivar) un DID y su DID-document asociado en una determinada cadena *blockchain*.

La organización W3C es la encargada de especificar un modelo común, un formato de URL y el conjunto de operaciones aplicables a DID y los documentos asociados a dichos DID.

Un DID es una cadena de texto con el siguiente formato:

Figura 2.1: Formato de una dirección DID.

- **Esquema del URL:** en nuestro caso "`did`".
- **Identificador del método del DID:** un método cualquiera llamado "example" (Actualmente ya existen varios métodos implementados, por ejemplo el de bitcoin "btcr")
- **Identificador específico del método DID:** hemos usado de ejemplo `123456789abcdefghi`.

Al acceder a la cadena del *blockchain* deberíamos ser capaces de encontrar el documento DID al que apunta nuestro DID, un ejemplo:

[2]Definición extraída de la especificación *Decentralized Identifiers (DIDs) v1.0 Core architecture, data model, and representations* realizada por W3C el 5 de marzo de 2020.

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    // used to authenticate as did:...fghi
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  "service": [{
    // used to retrieve Verifiable Credentials associated with the DID
    "id":"did:example:123456789abcdefghi#vcs",
    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

Figura 2.2: Formato básico de un documento DID.

- **context:** indica el estándar que se va a seguir en este documento DID.
- **id:** se trata del *string* DID del individuo u organización.
- **authentication:** comprende unas reglas que permiten verificar al individuo que se encuentra en posesión del DID en cuestión.
- **service:** indica algún método para contactar con el propietario del DID, puede ser un servicio web, una dirección de correo electrónico...

Este documento DID nos proporciona un "pasaporte online" que nos permitirá identificarnos en una red descentralizada.

2.3.2. Hyperledger Fabric

Para realizar nuestro trabajo es imprescindible tener una base de *blockchain* para la implementación. Por esta razón hemos decidido estudiar y usar *Hyperledger Fabric*. Se trata del *software opensource* más grande en la actualidad para desarrollar un *blockchain* privado que administre las transacciones entre empresas de manera más segura y eficiente.

Un aspecto fundamental de *Hyperledger Fabric* es su diseño **modular**, el cual nos proporciona una base abstracta a partir de la cual podemos implementar lo que necesitemos sin restricciones. Veremos más adelante en qué consisten dichos módulos.

Hyperledger Fabric no deja de ser un *blockchain* más, por lo que guarda un registro de toda la cadena con las transacciones pasadas. En el ejemplo de *Bitcoin* tenemos que dicho registro lleva todos los traspasos de monedas entre usuarios, en nuestro caso con *Hyperledger Fabric*, la definición es mucho más general pues lo que se traspasan son *assets*.

Más concretamente podemos definir un *asset* como una pareja **clave-valor** que permanece en la cadena y que además tiene un estado (por ejemplo a quién pertenece). Para modificar estos *assets*, se necesita el uso de un *chaincode* que será desarrollado por la implementación específica de cada proyecto. Todas las organizaciones que usen *Hyperledger Fabric* deben usar un *chaincode* común que indicará cómo se llevarán a cabo los *smart contracts* que se lanzarán a la red. En definitiva el *chaincode* se encarga de:

- Definir los *assets* y las instrucciones para modificarlos.
- Todos los miembros de la red del *blockchain* interactúan con ella mediante lo especificado en el *chaincode*.
- Define el protocolo de identificación de los miembros que administra sus ID y los autentifica.
- Acceder a las listas de permisos que da una capa más de seguridad.

En las redes de *Hyperledger Fabric* hay dos tipos de nodos en función de su actuación:

- **Peer Node:** son aquellos cuyo trabajo es la verificación y la ejecución del *chaincode* de las transacciones que se propagan a la red para ser añadidas a la cadena.

- **Orderer Node:** estos son los nodos que actúan como clientes, es decir son los que solicitan y propagan las transacciones.

Además en *blockchain* en sí se compone de dos partes fundamentales.

- **Blockchain Log:** lleva el historial de todas las transacciones que han sido admitidas a la cadena en bloques desde el comienzo del sistema.

- **Base de datos del estado:** para mejorar la eficiencia de cómo se encuentra el sistema en cada momento, hay una base de datos auxiliar que mantiene el estado de todas las transacciones. (Así no hay que recorrer todo el historial para saber cómo se encuentra un *asset* en concreto)

Nota: En el *blockchain* de Bitcon **no** existe una base de datos de este tipo, por lo que, en términos de eficiencia, es considerablemente peor que *Hyperledger Fabric*; ya que, para averiguar el estado de una cuenta (su saldo) es necesario calcular todas las transacciones que la involucran desde el comienzo de Bitcoin.

La clave fundamental para entender la necesidad de usar *Hyperledger Fabric* en nuestro trabajo es la siguiente: históricamente las empresas han rechazado el uso de tecnologías *blockchain* para administrar las transacciones entre las mismas por un problema en principio sin solución, **la falta de privacidad**. Esto se debe a que la cadena es pública para todos los participantes, por lo que las transacciones entre usuarios serían visibles para cualquiera.

Aquí llega la solución de *Hyperledger Fabric* con los **canales privados**, cuyo objetivo es:

- Restringir cómo los mensajes se asignan a la cadena para dar privacidad a ciertos subconjuntos de *assets*.
- Todos estos datos son invisibles para los usuarios que no tienen permiso.
- Permite que todas estas transacciones privadas se realicen con independencia de la cadena que tiene todas las públicas.

Para conseguir estas características los canales se implementan de la siguiente forma:

Antes que nada, los canales son subredes privadas entre miembros del *blockchain* público, de esta manera los miembros de un canal privado deben autentificarse pues ha de existir un consenso entre las partes que indique qué usuarios pueden unirse. De esta manera un canal conlleva una serie de nodos *peer* y nodos *orderer* privados junto con una cadena compartida.

La clave de la implementación reside en esta cadena compartida, que se trata de una cadena que comienza como una copia de la cadena pública, a partir de la cual se realizan transacciones que únicamente los miembros del canal podrán ver pues esta cadena compartida no se anuncia públicamente a todos los nodos *peer* sino que solo se anuncia a aquellos *peer* que sean miembros del canal en cuestión. Recordamos que para unirse a un canal privado hacen falta permisos generados por un *Membership Services Provider (MSP)*, el cual se decidió por convenio entre los miembros que crearon dicho canal.

Con toda esta información podemos extraer el siguiente resumen sobre *Hyperledger Fabric*

Características de *Hyperledger Fabric*

- Definir tipos de *assets* y el protocolo que los administra.
- Establecer los permisos sobre qué usuarios se pueden unir a la red.
- Dos tipos de nodos, *peer* y *orderer*.
- Cadena que consiste en un registro de transacciones y de una base de datos con el estado actual.
- La creación, eliminación y modificación de *assets* se realiza a través de *chaincode*.
- La existencia de canales privados que permitan crear copias secundarias de la cadena pública para poder realizar transacciones privadas únicamente entre miembros del canal.

2.4. Especificación

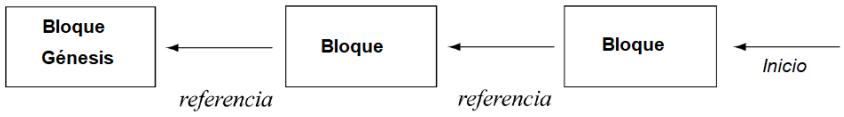
Con todo lo estudiado y teniendo en mente las conclusiones que obtuvimos durante el proceso análisis de la competencia, ya estamos listos para presentar la especificación del prototipo que vamos a implementar. Es importante señalar que la filosofía de la implementación va a depender en gran medida de la base que nos proporciona Hyperledger Fabric pues deberemos adaptarnos a ella. Para poder dar una visión completa del sistema vamos a ir definiendo las siguientes características:

Componentes de la prueba de concepto

- **Estructura del Blockchain**
  - Cadena
  - Bloques
  - Base de datos
  - Canales
- **Actores del sistema**
- **Contenido del wallet**
- **Chaincode**
  - Operaciones permitidas
- **Verificación de credenciales**
- **Entorno de ejecución**

Estructura del Blockchain

La **cadena** será una estructura de datos pública tal que todos los usuarios posean una copia de la misma. Dicha estructura es equivalente a una lista enlazada, cuyas piezas de enlace serán los **bloques**. Hay que indicar que el primer bloque de todos será especial y le denominaremos "bloque génesis", además es importante que la lista no es doblemente enlazada, esto quiere decir que tan solo se puede recorrer en una dirección: de bloque más reciente a bloque más antiguo. Más adelante veremos mecanismos para mejorar la eficiencia de las búsquedas. Para aclarar la idea de la estructura, tenemos la siguiente representación gráfica:



Definamos entonces la pieza fundamental de nuestra estructura de datos: los bloques. Un **bloque** consiste de las siguientes partes:

Cabecera

- **ID del bloque:** Se trata de un identificador único del bloque, una forma simple de asegurar dicha unicidad es atribuirle su posición en la cadena (concepto que se explica más adelante) como ID del bloque.
- **Hash de datos:** Una vez se haya configurado completamente un bloque, será necesario crear un hash criptográfico a partir de dicha información y añadirla en este campo. El hash de datos tiene como entrada todos los apartados del bloque excepto éste y el de metadatos.
- **Hash anterior:** Se trata de una copia idéntica del hash criptográfico que se asignó al bloque anterior de la cadena.

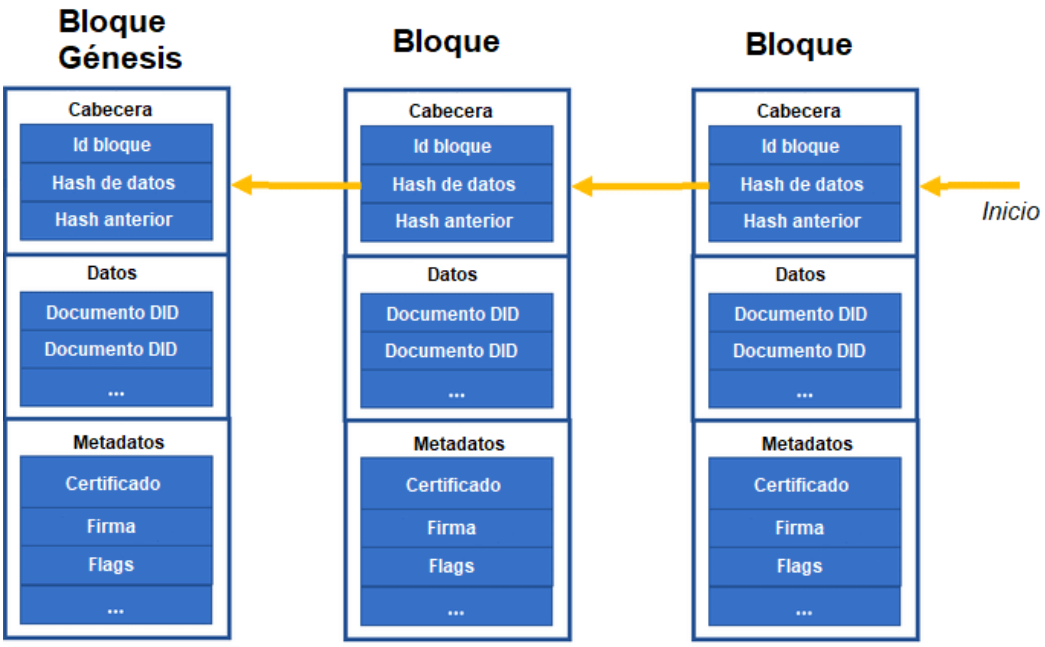
Datos

- **Documentos DID:** En esta sección se introducirá uno o varios documentos DID que vienen especificados tal y como los estudiamos en el apartado de estudio de herramientas auxiliares.

Metadatos

- **Certificado y firma:** El creador del bloque debe añadir su certificado y firma para que el resto de verificadores del sistema puedan validar el bloque, la relevancia de este apartado se mostrará más adelante.
- **Indicador de válido/inválido:** Se trata de una función auxiliar que permite detectar a los verificadores si el bloque está repetido.

Con esta estructura podemos ahora visualizar mejor la cadena:



Acabamos este apartado de estructura del *blockchain* presentando los dos últimos componentes:

- **Base de datos del estado:** como indicamos en la cadena, se trata de una lista enlazada en una dirección por lo que una búsqueda a los bloques más antiguos puede llegar a ser costosa con el tiempo. Para mejorar la eficiencia de los búsquedas de los documentos DID nuestro sistema implementa una base de datos de tipo *CouchDB* donde se guardan las conexiones (DID) $\Longleftrightarrow$

(Documento DID) de las operaciones de búsqueda más recientes. De esta manera solo habrá que recorrer la cadena en caso de que no se encuentre el documento en la base de datos.

- **Canales:** para obtener una copia de la cadena y poder introducir documentos DID en la misma, debe de establecer una conexión con el canal. El canal es esencialmente la red donde están conectados todos los miembros que participan en el sistema, a través de él se realizan las operaciones de introducción de nuevos bloques y verificación/publicación de la cadena actualizada.

Actores del sistema

Nuestra red descentralizada va a ser mantenida por dos tipos de participantes cuyas funcionalidades son distintas:

- **Usuarios:** son todos aquellos individuos u organizaciones que introducen sus respectivos documentos DID a la cadena. Recordemos que el documento DID contiene tanto una clave pública como un método de contacto, luego esto nos permitirá solucionar el problema de identificar al poseedor de la clave. Este tipo de actor corresponde con el definido como "orderer node" de Hyperledger Fabric.
- **Verificadores:** son aquellos individuos que se encuentran a la escucha de la aparación de bloques de nueva creación, su objetivo es la de validar que la sintaxis del bloque es correcta y de buscar el *hash* criptográfico que se introduce en el apartado *hash* de datos. Una vez encontrado un *hash* correcto, los verificadores envían mensajes de actualización con la nueva cadena. Nótese que dicho *hash* no es un cálculo trivial, pues usualmente se exigen ciertas características al mismo que solo se pueden resolver por fuerza bruta. Por ejemplo que en el final del *hash* haya N número de ceros.

Contenido del wallet

El aspecto clave de nuestro prototipo hemos visto que es el hecho de que en la cadena pública tan solo se pueden introducir documentos DID. Esto lo complementamos con que solo permitimos añadir identificaciones en el *wallet*, es decir una clave pública. Para inscribirse como usuario al sistema es necesario mandar una petición de ingreso a miembros ya existentes del mismo, en caso de aceptar al nuevo usuario, a éste se le asigna en su *wallet* un documento de texto firmado por un verificador. A partir de este momento, el usuario puede realizar operaciones de introducción de documentos DID en la cadena usando su clave pública. En caso de que una cuenta intente realizar una transacción a la cadena sin poseer el correspondiente documento firmado por un verificador en su *wallet*, entonces la transacción será invalidada por los verificadores.

Chaincode

Recordamos que para la unión de un nuevo usuario a la red era necesario una petición de ingreso, ahora bien, una vez dentro todos los usuarios deben ser capaces de ponerse de acuerdo en el conjunto de transacciones válidas. Para ello, en nuestro sistema desarrollaremos un *chaincode*, es decir, el conjunto de instrucciones con las funcionalidades que se permiten realizar en la red. Dicho *chaincode* debe ser aprobado por todos los usuarios del sistema que quieran usarlo, esto implica que son los usuarios los que deben asegurarse de que el *chaincode* realmente hace lo esperado. Una vez que el usuario ha ingresado en una red con su clave pública en el *wallet* y ha aceptado el código del *chaincode*, éste tendrá permitido llevar a cabo todas las funciones que le proporciona. Para nuestra prueba de concepto vamos a implementar las siguientes:

- **queryAllDID:** [Sin parámetros de entrada]

  Función que recorre toda la cadena pública y devuelve todos los DID-Document que pertenecen a la misma.

- **queryDID:** [Dirección DID como entrada]

  Dado un DID, es decir el análogo a dirección URL del DID (el ejemplo que vimos cuando estudiamos los DID era "`did:example:123456789abcdefghijk`"), busca en la cadena un DID-Document cuyo campo "`id`" coincida con el buscado.

- **addDIDdoc** [DID-Document como entrada]

  Para esta función el usuario proporciona un DID-Document y los verificadores actúan siguiendo el siguiente esquema:

  - Comprueba que el usuario que inicia la transacción lo hace usando una clave pública de su *wallet*.
  - Comprueba que la sintaxis del DID-Document es correcta de acuerdo con la especificación proporcionada por el W3C.
  - Para finalizar se realiza la prueba de trabajo, es decir, se genera el bloque correspondiente que va contender al documento y se hace el cálculo del *hash* de datos para poder insertarlo en cadena.
  - Una vez calculado el *hash* el verificador comienza a anunciar la nueva cadena para el resto de miembros de la red.

Verificación de credenciales

Recordemos que los DID-Document tienen dos secciones esenciales: "`authentication`" y "`service`". En la primera se dispone de algún mecanismo de encriptación, en nuestro caso usaremos claves RSA. Mientras que en la segunda sección que expone alguna forma de contacto del usuario, puede ser una dirección IP, una dirección de correo electrónico, incluso una dirección física si se trata de una organización... Esto nos resuelve el problema de asociar la relación usuario-clave. Es decir si pensamos en una dirección DID como un apodo y recuperamos su documento asociado desde la cadena, podemos asegurarnos al contactar con el medio que aparece en "service" que la clave pública pertenece al usuario con dicho apodo (puesto que en el nos responde demostrando que tiene posesión de la clave privada asociada).

Con todo esto ya podemos establecer una conexión con el usuario que posee dicha dirección DID y estar seguros de que es el individuo que esperábamos. Así pasamos a la última parte del trabajo y es la verificación de credenciales. Para este caso tendremos que acudir a la ayuda de una CA (*Certificate Authority*), sin embargo, tan solo será una vez por credencial. El proceso es el siguiente, el usuario pide a la CA que le certifique con su clave privada un credencial que antes ha tenido que presentar. Una vez obtiene el certificado, el usuario lo guarda para su uso posterior. Ahora cuando se realice la conexión segura siguiendo lo que hemos indicado anteriormente, el usuario podrá presentar su certificado y así acreditar el credencial en cuestión. Esta acreditación deberá llevarse a cabo mediante un protocolo auxiliar adicional y externo al prototipo propuesto, se trataría de añadir una forma estandariza de enviar los certificados firmados por la CA que posee el individuo. Para este trabajo no nos hemos centrado en desarrollar dicho protocolo pues recordamos que el problema principal a resolver es el de las relaciones usuario-clave pública, para ello nos hemos apoyado en la tecnología blockchain.

Entorno de ejecución

Todo el prototipo se va desarrollar en un sistema Linux, la versión 19.10 de Ubuntu. Vamos a partir de la base que nos proporciona Hyperledger Fabric, para ello usaremos el código *Open Source* que aparece en su GitHub oficial: *https://github.com/hyperledger/* . Ahora bien, La amplia gama de herramientas que proporciona el proyecto de Hyperledger está fuera del alcance del prototipo que vamos a desarrollar en este trabajo, por lo que nosotros nos vamos a restringir a trabajar con el entorno de ejemplos. Es decir nuestro proyecto comienza a partir de clonar los ficheros de *https://github.com/hyperledger/fabric-samples*

El proyecto Hyperledger posee una licencia Apache 2.0, por lo que tras documentar adecuadamente la fuente del mismo (más información en la sección de bibliografía), tenemos derecho a usar el código para nuestro prototipo.

En cuanto a los ejemplos que nos proporciona la carpeta `fabric-samples`, al final nos hemos decantado por usar como base el llamado "`FabCar`". Se trata de un prototipo de red descentralizada en la que se gestionan coches y sus propietarios. A pesar de que también consideramos en un principio basarnos en "`first-network`" que es una prueba de red de criptomonedas, las operaciones de `FabCar` nos parecieron más fácilmente generalizables.

Para entender la siguiente sección con la implementación vamos a presentar el esquema que sigue el prototipo `FabCar`:

- Se creará un canal con la cadena al que todas las máquinas ingresarán
  - `mychannel`
- Hay dos organizaciones y cada una tiene dos verificadores y una CA
  - `Org1`
    - `peer0.org1.example.com` *
    - `peer1.org1.example.com`
    - `ca_peerOrg1`
  - `Org2`
    - `peer0.org2.example.com` *
    - `peer1.org2.example.com`
    - `ca_peerOrg2`
- Existen 6 usuarios que realizan transacciones a la cadena
  - `orderer.example.com`
  - `orderer1.example.com`
  - `orderer2.example.com`
  - `orderer3.example.com`
  - `orderer4.example.com`
  - `orderer5.example.com`
- Hay cuatro bases de datos
  - `couchdb0`
  - `couchdb1`
  - `couchdb2`
  - `couchdb3`

* : Estos verificadores son además asignados como los MSP[3]de cada organización.

Una duda natural que surge es cuál será el papel de las máquinas `ca_peerOrg1` y `ca_peerOrg2`. Éstas se encargan de funcionalidades auxiliares:

- Ayudar a los MSP (en este caso `peer0.org1.example.com` y `peer0.org2.example.com`), en el proceso de ingreso de nuevos usuarios, generando los documentos firmados que se añadirán al *wallet* del nuevo miembro para que pueda realizar transacciones.
- Asegurar que solo los miembros registrados son capaces de realizar transacciones en la red, puesto que, cuando un usuario intente realizar una transacción las máquinas `ca_peerOrg1` y `ca_peerOrg2`

[3]Recordamos que un MSP era un *Membership Service Provider*, es decir los encargados de llevar a cabo el proceso que permite el ingreso de nuevos usuarios a la red.

comprobarán que dicho usuario tiene en su *wallet* una clave pública válida, es decir, obtenida durante el proceso usual de ingreso.

Durante la prueba del prototipo, nosotros tomaremos el papel de `orderer.example.com`, desde donde ingresaremos a la red y realizaremos una serie de transacciones.

Todo este esquema parece bastante complejo, así que presentamos la figura 2.3 con todas las máquinas:

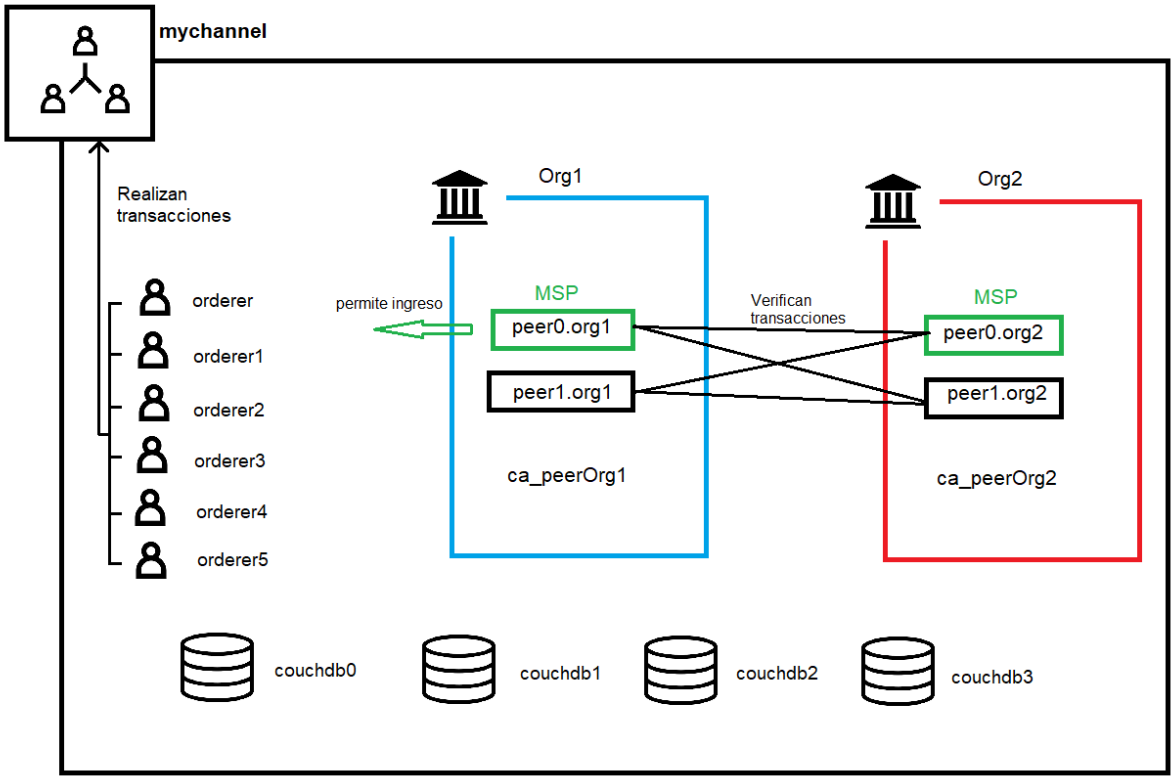


Figura 2.3: Esquema con todos los actores y elementos del prototipo que vamos a desarrollar.

2.5. Objetivos del trabajo e implementación

Nuestra tarea consistirá en:

- Alterar el *chaincode* localizado en "`/fabric-samples/chaincode/fabcar/go/fabcar.go`" de tal manera que la cadena pase a guardar DID-Documents en vez de coches y además redefinir todas las funciones que pueden usar las máquinas de la red para adaptarlas a nuestro objetivo.
- Crear en "`/fabric-samples/fabcar/javascript/`" los ficheros `.js` que los usuarios ejecutarán para realizar las operaciones sobre la cadena. (Estos `.js` harán llamadas a las funciones implementadas en el `.go` del *chaincode*)

Todos estos cambios se realizarán a partir de un *fork* del github con el proyecto oficial de Hyperledger Fabric. Se puede acceder a dicho proyecto con el código nuevo a través del siguiente enlace

https://github.com/pmartinhuertas/fabric-samples

2.5.1. Herramientas auxiliares

El primer paso de esta sección es poner en marcha todo el sistema base que nos proporciona Hyperledger Fabric. Para ello comenzamos por clonar el repositorio `hyperledger/fabric-samples`, a continuación tenemos que instalar todas las herramientas auxiliares de las que hace uso el prototipo.

- El primer paso será instalar todos los archivos binarios, los propios desarrolladores de `hyperledger` proporcionan un instalador automático.
- El segundo paso será instalar la herramienta `Docker`, la cual nos permitirá gestionar todos los procesos que lancemos en el prototipo mediante el uso de contenedores. Recordemos que el ejemplo tiene 4 verificadores, 6 clientes, 4 bases de datos y cada uno se ejecuta en un entorno distinto.
- El tercer paso consistirá en instalar el lenguaje `Go` en una versión 1.13 o posterior. En esencia es un lenguaje similar a `C`, su utilidad es que el *chaincode* del ejemplo `FabCar` que vamos a modificar se encuentra escrito en `Go` por lo que nos vamos a adaptar a él.
- El cuarto paso será instalar `Node.js`, veremos que las transacciones que ejecutan los usuarios serán ficheros `.js` (terminación de javaScript), los cuales ejecutaremos con esta herramienta.
- El quinto y último paso será instalar la versión de `Python 2.7`. Actualmente la versión de Ubuntu más reciente contiene la versión 3.5.1, sin embargo, para `Node.js SDK` algunas operaciones solo funcionarán correctamente con la versión 2.7.

Nota: Una guía más detallada del proceso de instalación de todas las herramientas necesarias se encuentra en las siguientes referencias de la bibliografía: [4] y [5].

2.5.2. Generación del entorno

El lanzamiento de nuestro prototipo se realizará navegando a la carpeta "`fabric-samples/fabcar/`" y ejecutando el script "`startFabric.sh`", a continuación se montarán todos los actores y elementos de nuestra red. Repasemos ayudándonos de figuras que contienen la salida del terminal que ejecuta el *script*, todos los pasos que se realizan cronológicamente para montar el sistema:

El primer paso de todos es tan solo auxiliar, pues su objetivo será la eliminación de todos los ficheros y los procesos que se hubiesen generado durante una ejecución anterior del sistema. Esto nos asegura evitar errores causados por el solapamiento de ejecuciones.

```
Removing cli                    ... done
Removing peer0.org2.example.com ... done
Removing peer1.org2.example.com ... done
Removing peer0.org1.example.com ... done
Removing peer1.org1.example.com ... done
Removing orderer5.example.com   ... done
Removing couchdb2               ... done
Removing ca_peerOrg2            ... done
Removing orderer.example.com    ... done
Removing couchdb0               ... done
Removing couchdb3               ... done
Removing orderer2.example.com   ... done
Removing orderer4.example.com   ... done
Removing couchdb1               ... done
Removing ca_peerOrg1            ... done
Removing orderer3.example.com   ... done
Removing network net_byfn
Removing volume net_orderer.example.com
Removing volume net_peer0.org1.example.com
Removing volume net_peer1.org1.example.com
Removing volume net_peer0.org2.example.com
Removing volume net_peer1.org2.example.com
Removing volume net_orderer2.example.com
Removing volume net_orderer3.example.com
Removing volume net_orderer4.example.com
Removing volume net_orderer5.example.com
Removing volume net_peer0.org3.example.com
WARNING: Volume net_peer0.org3.example.com not found.
Removing volume net_peer1.org3.example.com
WARNING: Volume net_peer1.org3.example.com not found.
```

Figura 2.4: Eliminación de todos los procesos y ficheros de la ejecución anterior.

Acto seguido, vamos a generar todos los ficheros de configuración.

- Generamos el bloque génesis, es decir, el primer bloque de la cadena. Recordemos que cada bloque debe contener el *hash* del anterior, sin embargo, para este primer bloque es necesario hacer una creación especial.
- Creamos el fichero de configuración "`channel.tx`". En él se especificarán las *config policies*, es decir las políticas de configuración que indicarán cuál será el proceso de creación/unión/interacción/salida del propio canal. Esto quiere decir que aquí se indicará quién en un principio tendrá los permisos de MSP.
- El último paso es añadir a este fichero fichero "`channel.tx`" los *anchor peer*, es decir vamos a indicarle a la configuración que tanto la organización 1 como la organización 2 tendrán permisos de MSP para poder decidir quién entra en el canal.

```
Generate CCP files for Org1 and Org2
/home/osboxes/fabric (copy)/first-network/../bin/configtxgen
##########################################################
#########  Generating Orderer Genesis block ##############
##########################################################
2020-03-30 05:59:56.526 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-03-30 05:59:56.667 EDT [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: etcdraft
2020-03-30 05:59:56.667 EDT [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" election_tick:10 heartbeat_tick:1 max
_inflight_blocks:5 snapshot_interval_size:16777216
2020-03-30 05:59:56.667 EDT [common.tools.configtxgen.localconfig] Load -> INFO 004 Loaded configuration: /home/osboxes/fabric (copy)/first-network/configtx.yaml
2020-03-30 05:59:56.672 EDT [common.tools.configtxgen] doOutputBlock -> INFO 005 Generating genesis block
2020-03-30 05:59:56.673 EDT [common.tools.configtxgen] doOutputBlock -> INFO 006 Writing genesis block

#################################################################
### Generating channel configuration transaction 'channel.tx' ###
#################################################################
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel
2020-03-30 05:59:56.786 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-03-30 05:59:56.926 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/osboxes/fabric (copy)/first-network/configtx.yaml
2020-03-30 05:59:56.926 EDT [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
2020-03-30 05:59:56.932 EDT [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
+ res=0
+ set +x

#################################################################
#######    Generating anchor peer update for Org1MSP   ##########
#################################################################
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP
2020-03-30 05:59:57.047 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-03-30 05:59:57.199 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/osboxes/fabric (copy)/first-network/configtx.yaml
2020-03-30 05:59:57.199 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
2020-03-30 05:59:57.203 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
+ res=0
+ set +x

#################################################################
#######    Generating anchor peer update for Org2MSP   ##########
#################################################################
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP
2020-03-30 05:59:57.292 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-03-30 05:59:57.428 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/osboxes/fabric (copy)/first-network/configtx.yaml
2020-03-30 05:59:57.428 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
2020-03-30 05:59:57.432 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
+ res=0
+ set +x
```

Figura 2.5: Creación de bloque génesis y generación de la configuración del canal "`mychannel`".

Lo siguiente es crear todos los procesos que comprenden a los usuarios, verificadores, bases de datos. Además podemos ver como dichos procesos se gestionan mediante los contenedores que proporciona la herramienta `Docker`.

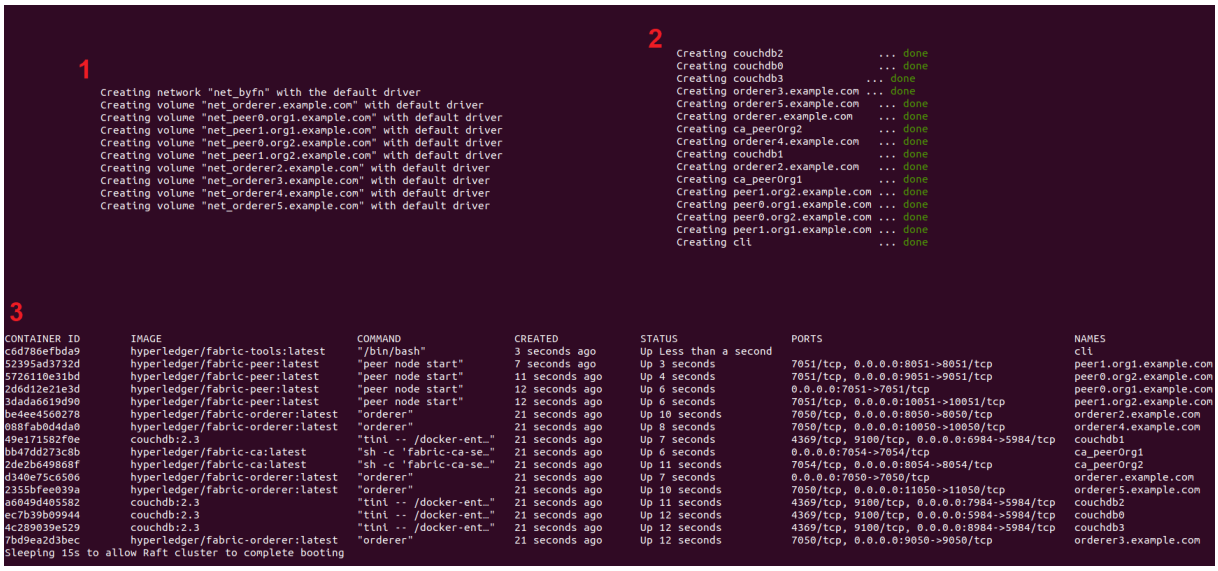


Figura 2.6: 1) Creación de drivers que permiten la conexión entre procesos. 2) creación de procesos. 3) Encapsulamiento de procesos en contenedores.

El siguiente paso es crear el único canal de nuestro prototipo que llevará el nombre de "`mychannel`". Para su creación se hace uso del fichero de configuración "`channel.tx`" que hemos generado anteriormente.

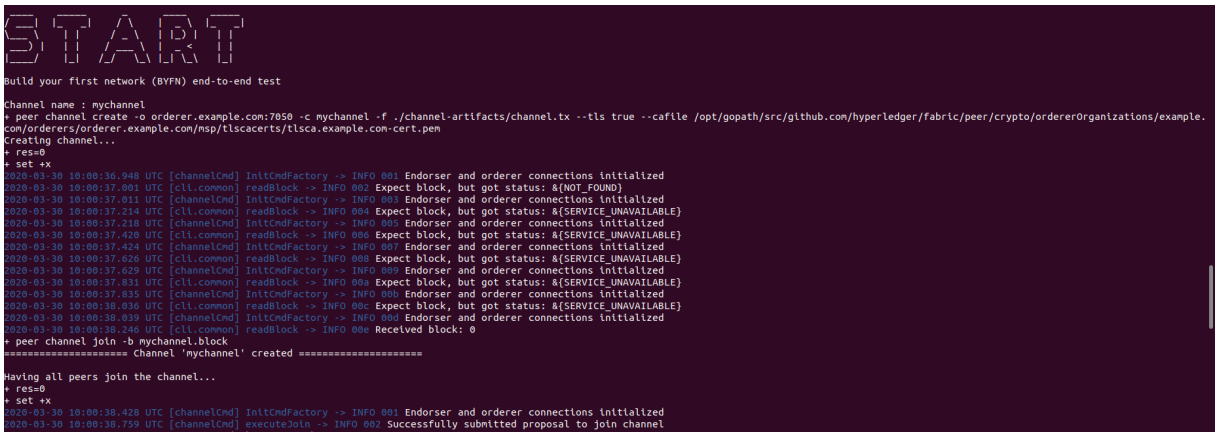


Figura 2.7: Creación del canal "`mychannel`".

Ahora vamos a llamar a todas las máquinas de las organizaciones (en nuestro caso procesos, porque estamos ejecutando todo en el mismo ordenador) a que se unan al canal común que contiene la cadena. Nótese que los 6 usuarios que hacen el papel de clientes no se unen al canal durante este proceso inicial, ellos deberán unirse más adelante pidiendo permiso a alguno de los dos MSP del sistema, uno de cada organización.

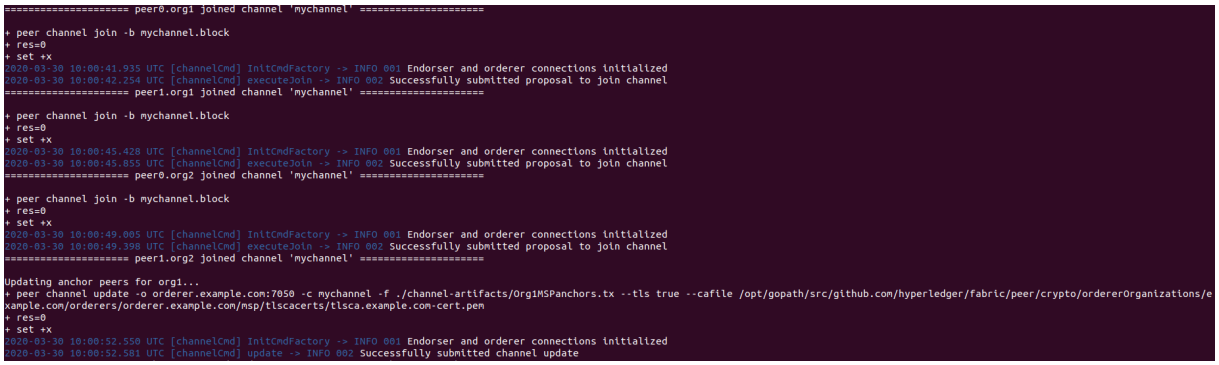


Figura 2.8: Unión de máquinas de organizaciones al canal "`mychannel`".

Finalmente, usando la configuración de los dos MSP que definimos en el canal, vamos a asignar a los MSP: tenemos el primero como `peer0.org1.example.com` y el segundo como `peer0.org2.example.com`.

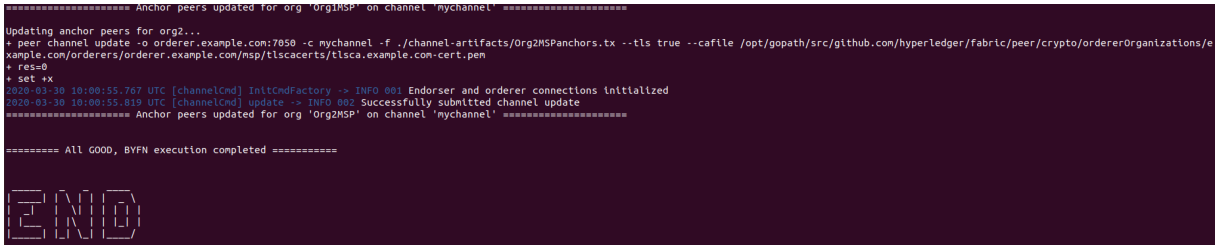


Figura 2.9: Asignación de las máquinas que actuarán como MSP en el canal.

El último paso del procedimiento de inicialización del entorno es el de aceptar el *chaincode*. Es decir, todas las máquinas que pertenezca a la red deben aprobar el *chaincode*; solo una vez aceptado podrán realizar transacciones sobre la misma (claramente usando únicamente las funciones que se definen en dicho *chaincode*).

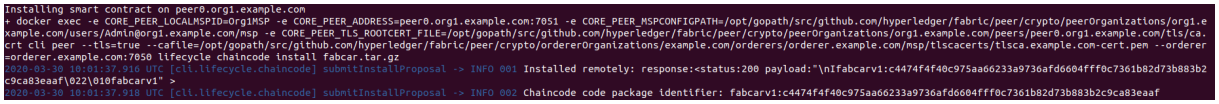


Figura 2.10: Ejemplo de instalación y aceptación del *chaincode* de la máquina `peer0.org1.example.com`.

Esto completa la generación del entorno del prototipo.

2.5.3. Implementación del chaincode

Ahora mostramos la modificación del fichero "`/fabric-samples/chaincode/fabcar/go/fabcar.go`" para su adaptación a soportar la gestión de documentos DID. El primer paso consiste en definir la estructura que contendrá los datos de dichos documentos, a pesar de que en la especificación proporcionada por W3C existe una amplia variedad de campos adicionales para rellenar en el documento, hemos decidido que los documentos de nuestro proyecto tengan únicamente los parámetros esenciales.

Recordamos que los dos apartados esenciales de los documentos DID eran por un lado *authentication* y por otro *service*, es decir, el primero se refiere al método criptográfico que se usará para la comunicación (en nuestro caso claves públicas RSA) y el segundo indica la forma de contacto del poseedor del documento DID que en los ejemplos será un sitio web, sin embargo, éste podría ser cualquier medio como un correo electrónico, una dirección física...

Pasemos entonces a las cuatro funciones que implementa el *chaincode* para manipular la cadena pública.

```
type Did struct {
    Id                          string `json:"id"`
    AuthenticationId            string `json:"authenticationId"`
    AuthenticationType          string `json:"authenticationType"`
    AuthenticationController    string `json:"authenticationController"`
    AuthenticationPublicKeyPerm string `json:"authenticationPublicKeyPerm"`
    ServiceId                   string `json:"serviceId"`
    ServiceType                 string `json:"serviceType"`
    ServiceEndPoint             string `json:"serviceEndPoint"`
}
```

Figura 2.11: Definición de la estructura que representa a un documento DID.

El código completo del fichero con el *chaincode* se puede obtener a partir del siguiente enlace:

https://github.com/pmartinhuertas/fabric-samples/blob/master/chaincode/fabcar/go/fabcar.go

- **InitLedger**
  - Parámetros de entrada: ninguno.
  - Funcionalidad: se le llama únicamente cuando se crea la cadena por primera vez, su objetivo es añadir un par de DID de ejemplo a la cadena inicial que nos servirán más tarde para las pruebas de ejecución que realicemos sobre el código nuevo.
  - Devuelve: nada en caso de funcionar correctamente y un mensaje de error si falla la inserción de los documentos de ejemplo.

Estos son los documentos de ejemplo:

```
Did{Id: "did:example:12346789abcdefghi", AuthenticationId: "did:example:12346789abcdefghi#keys-1",
    AuthenticationType: "RsaVerificationKey2018", AuthenticationController: "did:example:12346789abcdefghi",
    AuthenticationPublicKeyPerm: "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n",
    ServiceId:                   "did:example:12346789abcdefghi#vcs", ServiceType: "VerifiableCredentialService",
    ServiceEndPoint: "https://example.com/vc/"},

Did{Id: "did:example:12346789asdfghjkl", AuthenticationId: "did:example:12346789asdfghjkl#keys-1",
    AuthenticationType: "RsaVerificationKey2018", AuthenticationController: "did:example:12346789asdfghjkl",
    AuthenticationPublicKeyPerm: "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n",
    ServiceId:                   "did:example:12346789aasdfghjkl#vcs", ServiceType: "VerifiableCredentialService",
    ServiceEndPoint: "https://example2.com/vc/"},
```

Figura 2.12: Documentos DID de ejemplo en la cadena inicial.

- **CreateDid**
  - Parámetros de entrada: uno por cada atributo definido en struct Did más un atributo auxiliar de clave (*key*).
  - Funcionalidad: dados todos los datos de un documento DID, genera la estructura que lo representa y lo introduce en la cadena pública.
  - Devuelve: nada.
- **QueryDidByKey**
  - Parámetros de entrada: cadena que representa la clave.
  - Funcionalidad: dada una clave (por ejemplo “DID1”) busca en la cadena al documento DID al que apunta y lo devuelve.
  - Devuelve: si se encuentra devuelve un puntero al documento DID, en caso contrario muestra un mensaje de error por pantalla.

- **QueryDidById**
  - Parámetros de entrada: cadena que representa la dirección de un documento DID.
  - Funcionalidad: dada una dirección de un documento DID (por ejemplo "`did:example:123456789abcdefghi`") busca en la cadena al documento DID al que apunta y lo devuelve.
  - Devuelve: si se encuentra devuelve un puntero al documento DID, en caso contrario muestra un mensaje de error por pantalla.
- **QueryAllDids**
  - Parámetros de entrada: ninguno.
  - Funcionalidad: recopila todos los documentos DID que existen en la cadena mediante el recorrido de todas las claves válidas.
  - Devuelve: estructura de diccionario con: clave (*key*) y valor (documento DID).

Cabe indicar una vez más que éstas serán las únicas operaciones permitidas sobre la cadena, puesto que todos los miembros de nuestra red descentralizada deberán aceptar solamente este *chaincode* antes de empezar a realizar transacciones sobre el sistema.

2.5.4. Implementación de los *scripts* de transacciones

Por un lado hemos visto que el *chaincode* proporciona las operaciones que se le permite realizar al usuario, sin embargo, el usuario es totalmente libre de implementar en cualquier lenguaje funciones que llamen a las del *chaincode* para llevar a cabo las transacciones deseadas. En este trabajo hemos realizado cuatro ficheros javascript que permitirán al usuario realizar las operaciones sobre la cadena de manera cómoda.

Nota: Al igual que antes, dejamos a continuación el enlace con la carpeta que contiene los 4 scripts.

https://github.com/pmartinhuertas/fabric-samples/tree/master/fabcar/javascript

Indicamos que la forma de ejecutar estos scripts es mediante una llamada en un terminal del tipo:

node nombre_fichero.js [argumentos]

Hagamos un repaso de los scripts que ofrecemos en nuestro prototipo:

- **enrollAdmin.js**
  - Argumento: ninguno.
  - Funcionalidad: realiza una petición a uno de los *Membership Service Provider* de la red pidiendo unirse con permisos de administrador, esto quiere decir añadir en "`/fabric-samples/fabcar/javascript/wallet`" un fichero llamado "`admin.id`" cuyo contenido será un certificado X.509 que permitirá identificarse como usuario administrador . Debido a la configuración base que se implementa en el *script* de inicio **startFabric.sh**, la transacción será correcta desde la máquina de nuestro prototipo para poder así realizar las pruebas oportunas.
- **registerUser.js**
  - Argumento: ninguno.
  - Funcionalidad: realiza una petición a uno de los *Membership Service Provider* de la red pidiendo unirse con permisos de usuarios usual, esto quiere decir añadir en "`/fabric-samples/fabcar/javascript/wallet`" un fichero llamado "`user1.id`" cuyo contenido será un certificado X.509 que permitirá identificarse como usuario del canal. En nuestro caso, este permiso será otorgado por poseer permisos de administrador.

- **QueryAllDids.js**
  - Argumento: ninguno.
  - Funcionalidad: en caso de ser miembro de una red y estando conectado a un canal de la misma, se realiza una búsqueda en la cadena pública que la recorre devolviendo una lista con todos los documentos DID que la componen.
- **QueryDidByKey.js**
  - Argumento: una clave.
  - Funcionalidad: en caso de ser miembro de una red y estando conectado a un canal de la misma, se realiza una búsqueda en la cadena pública filtrando por la clave proporcionada como parámetro. En caso de encontrar un documento DID con dicha clave, se devolverá el mismo por pantalla.
- **QueryDidById.js**
  - Argumento: una dirección DID.
  - Funcionalidad: en caso de ser miembro de una red y estando conectado a un canal de la misma, se realiza una búsqueda en la cadena pública filtrando por la dirección DID proporcionada como parámetro. En caso de encontrar un documento DID con dicha dirección DID, se devolverá el mismo por pantalla.
- **createDid.js**
  - Argumentos: los 8 campos que componen un documento DID básico y una clave válida.
  - Funcionalidad: proporcionados todos los campos que componen un documento DID, se genera una estructura que lo represente. Acto seguido se hace una petición de introducción del nuevo documento a los verificadores del canal y tras la prueba de trabajo se añade el documento al final de la cadena.

2.5.5. Prueba de ejecución

En este último apartado del trabajo, vamos a llevar a cabo paso a paso una serie de ejecuciones del *chaincode* que hemos implementado mediante las llamadas a los *scripts* `.js` . Los objetivos serán aclarar todas las dudas sobre el funcionamiento de la prueba de concepto y llevar a cabo los casos de usos expuestos en la sección 2.2. De aquí en adelante vamos a suponer que nos encontramos justo después del momento en el que acabamos la generación del entorno de la sección 2.5.2.

- **Registro al canal como administrador**

  Comenzamos por navegar a la carpeta "`/fabric-samples/fabcar/javascript/` " y ejecutamos la instrucción:

  node enrollAdmin.js

El resultado por pantalla de la transacción se muestra en la siguiente figura:

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node enrollAdmin.js
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Successfully enrolled admin user "admin" and imported it into the wallet
```

Figura 2.13: Inscripción como administrador.

Como podemos observar se ha añadido un fichero a nuestro *wallet* con los crendeciales necesarios para acreditarnos como administrador del canal. Ésto se ha hecho a través de la creación del fichero "admin.id".

Open admin.id ~/fabric/fabcar/javascript/wallet Save


```
{"credentials":{"certificate":"-----BEGIN CERTIFICATE-----

\nMIICATCCAaigAwIBAgIUEv6elngIK8RLJk43Pfuy/mX44p0wCgYIKo
ZIzj0EAwIw\nczELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3Ju
aWExFjAUBgNVBAcTDVNh\nbiBGcmFuY2lzY28xGTAXBgNVBAoTEG9yZz
EuZXhhbXBsZS5jb20xHDAaBgNVBAMT\nE2NhLm9yZzEuZXhhbXBsZS5j
b20wHhcNMjAwNDE5MjEzMTAwWhcNMjEwNDE5MjEz\nNjAwWjAhMQ8wDQ
YDVQQLEwZjbGllbnQxDjAMBgNVBAMTBWFkbWluMFkwEwYHKoZI\nzj0C
AQYIKoZIzj0DAQcDQgAER5mTkhJnbhLARp1Bxnd4M6kxHfWJuqIVGHqK
1d0v\nHxlH0MLg30rsPw+KWCaxcRM1YrE9f4e4lZgif1Qh0dqSJ6NsMG
owDgYDVR0PAQH/\nBAQDAgeAMAwGA1UdEwEBwQCMAAwHQYDVR0OBBYEF
N2q0OjF+YMP34hy2BwpmFBa\nkKXEMCsGA1UdIwQkMCKAIFkRPJMe8ev
ITl6Skm990XEihMgQHbFx06LxY1s5Vt4Y\nMAoGCCqGSM49BAMCA0cAM
EQCIEurq7ZFUK+Dq8iKtIh+kJ0daNWMa1bE8jkRUCj\nBygSAiBZ3ndb
2+6zy8vXznpiisKoas5LmUkUVpoDQCRsrWmN9Q==\n
-----END CERTIFICATE-----\n",

"privateKey":"-----BEGIN PRIVATE KEY-----
\r\nMIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgJq97
5HCOImY+Mkq\r\nWNbOfwEkAY2oukYt3dpxZAMXhIihRANCAARHmZOSE
mduEsBGnUHGd3gzqTEd9Ym6\r\nohUYeorV3S8fGUfQwuDfSuw/D4pYJ
rFxEzVisT1/h7iVmCJ/VCHR2pIn\r\n
-----END PRIVATE KEY-----\r\n"},

"mspId":"Org1MSP","type":"X.509","version":1}
```


Figura 2.14: Contenido el fichero "`admin.id`".

- **Registro al canal como usuario**

  El procedimiento es análogo al de registro como administrador, bastará ejecutar la instrucción:

  node registerUser.js

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node registerUser.js
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Successfully registered and enrolled admin user "user1" and imported it into the wallet
```

Figura 2.15: Inscripción como administrador usuario.

En este caso generaremos un fichero "`user1.id`", su contenido lo omitimos pues sigue el mismo formato que el mostrado en la ejecución anterior.

- **Obtención de todos los documentos DID de la cadena**

  Antes de realizar operaciones sobre la cadena, vamos a llevar a cabo una búsqueda de todos los documentos DID existentes inicialmente. Si recordamos nuestra implementación del *chaincode*, acudiendo a la figura 2.12 podemos ver que deberíamos obtener únicamente los dos documentos DID que especificamos en dicha sección. Para imprimir toda la cadena ejecutaremos:

node queryAllDids.js

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node queryAllDids.js
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Transaction has been evaluated, result is: [{"Key":"DID0","Record":{"id":"did:example:12346789abcdefg
hi","authenticationId":"did:example:12346789abcdefghi#keys-1","authenticationType":"RsaVerificationKe
y2018","authenticationController":"did:example:12346789abcdefghi","authenticationPublicKeyPerm":"----
-BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n","serviceId":"did:example:12346789abcdefghi#vcs","service
Type":"VerifiableCredentialService","serviceEndPoint":"https://example.com/vc/"}},{"Key":"DID1","Reco
rd":{"id":"did:example:12346789asdfghjkl","authenticationId":"did:example:12346789asdfghjkl#keys-1","
authenticationType":"RsaVerificationKey2018","authenticationController":"did:example:12346789asdfghjk
l","authenticationPublicKeyPerm":"-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n","serviceId":"did:e
xample:12346789aasdfghjkl#vcs","serviceType":"VerifiableCredentialService","serviceEndPoint":"https:/
/example2.com/vc/"}}]
```

Figura 2.16: Devolución por pantalla de todos los documentos DID de la cadena.

En efecto, después de alrededor de 3 segundos tras realizar la transacción, nos respondió un miembro del canal con los dos documentos que indicamos anteriormente.

- **Obtención de un documento DID con su dirección**

  Ahora vamos a comprobar que funciona la búsqueda de un documento concreto a través de su dirección. Para ello vamos a buscar el primer documento mediante la siguiente instrucción:

  node queryDidById.js did:example:12346789abcdefghi

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node queryDidById.js did:example:12346789abcdefghi
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"id":"did:example:12346789abcdefghi","authenticationId":"
did:example:12346789abcdefghi#keys-1","authenticationType":"RsaVerificationKey2018","authenticationCo
ntroller":"did:example:12346789abcdefghi","authenticationPublicKeyPerm":"-----BEGIN PUBLIC KEY...END
PUBLIC KEY-----\r\n","serviceId":"did:example:12346789abcdefghi#vcs","serviceType":"VerifiableCredent
ialService","serviceEndPoint":"https://example.com/vc/"}
```

Figura 2.17: Devolución por pantalla del documento DID buscado por su dirección.

- **Búsqueda de un documento DID con una dirección inexistente**

  Simplemente para mostrar que los miembros del canal son capaces de gestionar mensajes de error, vamos a ejecutar una búsqueda como la anterior pero esta vez le vamos a proporcionar una dirección errónea.

  node queryDidById.js did:example:wrong

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node queryDidById.js did:example:wrong
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Failed to evaluate transaction: Error: did:example:wrong does not exist
```

Figura 2.18: Devolución por pantalla del error de búsqueda del documento DID inexistente.

- **Obtención de un documento DID con su key**

  Este apartado es similar al anterior, a diferencia de que esta vez lo haremos usando una clave y buscaremos el segundo documento DID.

  node queryDidByKey.js DID1

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node queryDidByKey.js DID1
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"id":"did:example:12346789asdfghjkl","authenticationId":"
did:example:12346789asdfghjkl#keys-1","authenticationType":"RsaVerificationKey2018","authenticationCo
ntroller":"did:example:12346789asdfghjkl","authenticationPublicKeyPerm":"-----BEGIN PUBLIC KEY...END
PUBLIC KEY-----\r\n","serviceId":"did:example:12346789aasdfghjkl#vcs","serviceType":"VerifiableCreden
tialService","serviceEndPoint":"https://example2.com/vc/"}
```

Figura 2.19: Devolución por pantalla del documento DID buscado por su clave.

- **Introducción de un nuevo documento DID**

  Un aspecto esencial de la red es la introducción de nuevos documentos DID y asegurar que los verificadores del canal están funcionando correctamente generando los *hash* adecuados para añadir el bloque nuevo. En el *script* que vamos a ejecutar a continuación hemos introducido un documento de prueba para que sea añadido a la cadena.

node createDid.js

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node createDid.js
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Transaction has been submitted
```

Figura 2.20: Devolución por pantalla del éxito de introducción del nuevo documento DID.

- **Comprobación de introducción del nuevo documento DID**

  Tras un periodo de tiempo de no más de 1 minuto, los verificadores deben de haber sido capaces de verificar el bloque para ser añadido al final de la cadena. Una vez hemos realizado esta espera vamos a volver a hacer una nueva búsqueda con el contenido de toda la cadena para asegurarnos de que el nuevo documento DID ha sido introducido.

node queryAllDids.js

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node queryAllDids.js
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Transaction has been evaluated, result is: [{"Key":"DID0","Record":{"id":"did:example:12346789abcdefg
hi","authenticationId":"did:example:12346789abcdefghi#keys-1","authenticationType":"RsaVerificationKe
y2018","authenticationController":"did:example:12346789abcdefghi","authenticationPublicKeyPerm":"----
-BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n","serviceId":"did:example:12346789abcdefghi#vcs","service
Type":"VerifiableCredentialService","serviceEndPoint":"https://example.com/vc/"}},{"Key":"DID1","Reco
rd":{"id":"did:example:12346789asdfghjkl","authenticationId":"did:example:12346789asdfghjkl#keys-1","
authenticationType":"RsaVerificationKey2018","authenticationController":"did:example:12346789asdfghjk
l","authenticationPublicKeyPerm":"-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n","serviceId":"did:e
xample:12346789aasdfghjkl#vcs","serviceType":"VerifiableCredentialService","serviceEndPoint":"https:/
/example2.com/vc/"}},{"Key":"DID2","Record":{"id":"did:example:new","authenticationId":"did:example:n
ew#keys-1","authenticationType":"RsaVerificationKey2018","authenticationController":"did:example:new"
,"authenticationPublicKeyPerm":"-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n","serviceId":"did:exa
mple:new#vcs","serviceType":"VerifiableCredentialService","serviceEndPoint":"https://exampleNew.com/v
c/"}}]
```

Figura 2.21: Devolución por pantalla de todos los documentos DIDs de la nueva cadena.

- **Búsqueda del nuevo documento DID**

  Terminamos con la última instrucción de nuestra prueba, en ella vamos a buscar específicamente el nuevo documento DID comprobando así que la búsqueda indexada funciona correctamente.

node queryDidById.js did:example:new

```
osboxes@osboxes:~/fabric/fabcar/javascript$ node queryDidById.js did:example:new
Wallet path: /home/osboxes/fabric/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"id":"did:example:new","authenticationId":"did:example:ne
w#keys-1","authenticationType":"RsaVerificationKey2018","authenticationController":"did:example:new",
"authenticationPublicKeyPerm":"-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n","serviceId":"did:exam
ple:new#vcs","serviceType":"VerifiableCredentialService","serviceEndPoint":"https://exampleNew.com/vc
/"}
```

Figura 2.22: Devolución por pantalla de la búsqueda del nuevo documento DID usando su dirección.

Capítulo 3

Conclusiones

3.1. Resumen del trabajo y conclusión

Finalmente, tras alrededor de 30 páginas repletas de información, el contenido del trabajo sea, probablemente, demasiado "denso". La idea de este apartado va a ser un repaso breve que haga un seguimiento cronológico de todo lo que hemos visto, haciendo hincapié en los puntos clave que ayudan a entender la cadena de razonamiento del trabajo.

Propósito del trabajo: **identidades parciales**

Permitir la verificación de credenciales de manera selectiva a través de internet, es decir, el usuario es el que decide qué datos va a aportar. Para lograrlo hay que garantizar que dicho proceso de verificación es seguro. Éste es el punto en el que nos hemos centrado, la creación de un sistema base que proporcione dicha seguridad.

Las razones por las que se decidió que este trabajo era interesante desde el punto de vista de la investigación se pueden resumir en los siguientes puntos.

- **El trabajo trata un problema real** que existe en las *Certificate Authority* actuales. Cuando se proporciona una clave pública a un tercero, éste puede obtener todos los datos asociados a la misma, mientras que el usuario posiblemente solo quiera otorgarle cierta información.
- **Es un tema de investigación novedoso**, la prueba está en que uno de los máximos exponentes de este concepto *Rebooting Web of Trust* comenzó a finales de 2015.
- **En los últimos años ha surgido un auge de investigación sobre este tema**, esto se puede ver en proyectos de gran calibre como el desarrollado por IBM que actualmente se encuentra en una versión alfa cerrada.
- **Permite demostrar conocimientos de ingeniería informática** de multitud de ramas, pudiendo destacar:
  - Metodología de Ingeniería del Software realizando un análisis de la competencia y un apartado de casos de uso.
  - Conocimientos de programación tanto de lenguajes orientados a objetos (*javascript*) como de lenguajes imperativos (`Go` un lenguaje similar a `C`).
  - Dominio de conceptos de redes, centrados principalmente en criptografía de claves y certificados.
  - Ampliación en áreas avanzadas como la de redes descentralizadas del tipo *blockchain*.

Todos estos puntos reforzaron la idea de que era un trabajo que merecía la pena desde un punto de vista académico e incluso, también, desde un punto de vista comercial.

Para entender el desarrollo del trabajo es esencial entender los dos problemas fundamentales que nos plantean las identidades parciales:

- ¿Cómo podemos asociar una clave pública con un usuario concreto?
- ¿Cómo se realiza la verificación de credenciales?

A priori no eramos capaces de dar respuestas a estas preguntas por lo que el primer paso fue realizar un estudio exhaustivo de la competencia en proyectos similares para poder así extraer conclusiones. En el trabajo al final nos decantamos por dar solución a la primera de las preguntas pues abarcar ambas suponía una carga excesiva. Principalmente aprendimos que dicha conexión de clave pública-usuario antiguamente se realizaba mediante redes de confianza PGP; sin embargo, existe una alternativa más atractiva: redes descentralizas con *blockchain*. El otro punto clave es que va a ser necesario la ayuda de CA tradicionales mínimamente por lo que no es un gran problema. La idea es la siguiente, una vez se ha resuelto el problema de clave pública-usuario podemos garantizar que la comunicación es con quien realmente dice ser, por lo que el último paso sería que el usuario posea una serie de documentos firmados, cada uno verificando un credencial. Luego bastaría con mandar únicamente el certificado deseado durante la comunicación. Ciertamente la obtención de dicho documentos firmados supondría tener que acudir a una CA u otra organización de confianza, sin embargo consideramos éste un problema menor en comparación con asegurar que la conexión es segura gracias a la asociación usuario-clave pública.

Teniendo en cuenta todo lo indicado anteriormente pasamos al desarrollo en sí del trabajo, los dos conceptos fundamentales que hay que entender son:

- **Documento DID:** es un estándar que permite la identificación de usuarios (direcciones DID) con un servicio (forma de contacto, por ejemplo el correo electrónico). La idea es la siguiente: consideremos el pasaporte de un país, el tamaño, los campos, el estilo de letra, etc. Todos esos parámetros son un convenio que el gobierno del país en cuestión ha llevado a cabo. Estamos ante un fenómeno similar, necesitamos un documento que nos identifique pero para facilidad de tratamiento de los datos es imperativo que todos los usuarios usen la misma sintaxis. Exactamente eso define un documento DID.
- **Red descentralizada:** nos referimos a redes que funcionan mediante *blockchain*, es decir, nadie es el propietario absoluto de la red, sino que existen una serie verificadores de distintos organismos que soportan la red mediante "pruebas de trabajo".

Con estos dos conceptos ya podemos entender el objetivo de implementación del trabajo, vamos a crear una red descentralizada en la cual se puedan introducir documentos DID. De esta manera permitimos resolver el problema de relacionar clave pública con usuario y aseguramos que nadie tiene el control absoluto del medio. Tal sistema sería una solución para la primera parte del proyecto de identidades parciales, la segunda mitad es la verificación de credenciales que dejamos como una posible ampliación futura del proyecto. Puesto que hoy en día ya existen proyectos interesantes que podemos usar como base, nuestro código es una modificación de un sistema de *blockchain open source* llamado **Hyperledger Fabric**, en el cual adaptamos una red descentralizada que permitía la gestión de coches a una que permita la gestión de documentos DID. La implementación se resume en:

Implementación

- Alterar el *chaincode* en lenguaje `Go` de tal manera que la cadena pase a administrar documentos DID.
- Crear los ficheros javascript que los usuarios ejecutarán para realizar las operaciones sobre la cadena. Éstos ficheros son los que hacen llamadas al *chaincode*.

El proyecto está publicado en el enlance: https://github.com/pmartinhuertas/fabric-samples

Summary and Conclusion

Finally, after around 30 pages full of information, the content of the work is probably too "dense". The idea of this section is going to be a brief review that makes a chronological follow-up of everything we have seen, emphasizing the key points that help to understand the chain of reasoning of the work.

Goal of this Work: **Partial Identities**

To allow the verification of credentials selectively through the internet, that is: the user is the one who decides what data to provide. To achieve this, it must be ensured that said verification process is safe. This is the point on which we have focused, the creation of a base system that provides such security.

The reasons why this work was decided to be worthwhile from the research point of view can be summarized in the following points.

- **The work addresses a real problem** that exists in today's certification authorities. When a public key is provided to a third party, such third party can obtain all the data associated with the public key whereas there is the possibility that the user may only want to provide certain information.
- **It is a new research topic**, the proof is that one of the maximum exponents of this concept Rebooting Web of Trust began in late 2015.
- **In recent years there has been a boom in research on this topic**, this can be seen in large-scale projects such as the one developed by IBM, which is currently in a closed alpha version.
- **It allows to demonstrate computer engineering knowledge** of many branches, being able to highlight::
  - Software Engineering Methodology carrying out a competition analysis and a use cases section.
  - Knowledge of programming both object-oriented languages (`javascript`) and imperative languages (`Go` a language similar to `C`).
  - Expansion in advanced areas such as decentralized blockchain networks.

All these points reinforced the idea that it was worthwhile work from an academic point of view and even from a commercial point of view.

To understand the development of the work it is essential to understand the two fundamental problems that partial identities pose to us:

- ¿How can we associate a public key with a specific user?
- ¿How is Credential Verification Performed?

At first we were not able to provide answers to these questions, so the first step was to carry out an exhaustive study of the competition in similar projects in order to draw conclusions. For this particular work at the end we opted to give a solution to the first of the questions since covering both was an excessive amount of research. Mainly we learned that this public-user key connection was formerly made through trusted PGP networks; however, there is a more attractive alternative: decentralized networks with blockchain. The other key point is that traditional CA help is going to be minimally needed so it's not a big problem. The idea is as follows, once the public-user key problem has been solved, we can guarantee that communication is secure so the last step would be for the user to have a series of signed documents, each one verifying a credential. Then it would be enough to send only the desired certificate during the communication. Obtaining such signed documents would certainly mean having to go to a CA

or other trusted organization, however we consider this a minor problem compared to ensuring that the connection is secure thanks to the user-public key association.

Taking into account everything indicated above, we move on to the development of the work itself, the two fundamental concepts that must be understood are:

- **DID document:** it is a standard that allows the identification of users (DID addresses) with a service (contact form, for example email). The idea is as follows: Let's consider the size of a passport, the font style, the number of fields... All these parameters are a convention that the government of the country in question has imposed. We are facing a similar phenomenon, we need a document that identifies us but for ease of data processing it is imperative that all users use the same syntax. That's the philosophy behind the concept of DID document.
- **Decentralized network:** we refer to networks that work through blockchain, that is: nobody is the absolute owner of the network but there are a series of verifiers from different organizations that support the network through "proof of work".

With these two concepts we can already understand the goal of what we want to implement for this work: we are going to create a decentralized network in which DID documents can be entered. In this way we allow us to solve the problem of relating public key with user and we ensure that nobody has absolute control of the medium. Such system would be a solution to the first part of the partial identities project, the second half being the verification of credentials which we leave as a possible future upgrade of the project. As of today there are already interesting projects that we can use as a base, our code is a modification of an open source blockchain system called Hyperledger Fabric in which we adapted a decentralized network that allowed car management to one that allows management of DID documents. Implementation consists of:

Implementation

- Alter the chaincode written in `Go` language in such a way that the chain allows the management of DID documents.
- Create the javascript files that users will execute to perform operations on the chain. These files are the ones that make calls to the chaincode.

  The proyect is public through the link: https://github.com/pmartinhuertas/fabric-samples

3.2. Ampliación futura

Lo que hemos implementado durante este trabajo no es más que una prueba de concepto que nos permita mostrar el interés y la validez del tema que hemos tratado. Esto nos da una enorme libertad en cuanto a todo tipo de ampliaciones que podemos hacerle al prototipo. En caso de que el lector desee seguir trabajando sobre el proyecto o simplemente tenga curiosidad sobre cuáles son los pasos siguientes para mejorar el prototipo, a continuación presentaremos un listado con los posibles objetivos más relevantes que no se han llegado a tratar.

- **Permitir documentos DID con sintaxis general**

  Recordemos cómo eran los documentos DID que permitíamos en nuestro prototipo observando una figura que ya presentamos.

```
type Did struct {
    Id                          string `json:"id"`
    AuthenticationId            string `json:"authenticationId"`
    AuthenticationType          string `json:"authenticationType"`
    AuthenticationController    string `json:"authenticationController"`
    AuthenticationPublicKeyPerm string `json:"authenticationPublicKeyPerm"`
    ServiceId                   string `json:"serviceId"`
    ServiceType                 string `json:"serviceType"`
    ServiceEndPoint             string `json:"serviceEndPoint"`
}
```

  Figura 3.1: Estructura del documento DID en nuestro proyecto.

  Podemos fijarnos en que tan solo permitimos documentos DID básicos. Con esto nos referimos a que por ejemplo un documento DID según el estándar W3C puede tener más de un método de autenticación o más de un método de conexión, mientras que solo permitimos uno. Así que la tarea sería generalizar la sintaxis de estos documentos de manera que el sistema sea capaz de introducir documentos DID más complejos.

- **Crear un protocolo de comunicación auxiliar para verificar credenciales**

  Para este punto hay que entender qué problema resuelve nuestro prototipo. La red descentralizada con documentos DID nos permite establecer la conexión entre un usuario y una clave pública, es decir asociamos una dirección DID que actúa como seudónimo a un método de comunicación como por ejemplo una dirección de correo electrónico. Una vez resuelto el problema de identificar con quién nos estamos comunicando, es necesario tratar el problema de la verificación de credenciales. La solución que proponemos desde este trabajo es indicar una dirección IPv4/IPv6 en el campo *service* del documento DID e implementar un cliente-servidor para establecer una comunicación con el objetivo de que el usuario proporcione un certificado obtenido mediante una CA que le verifique un credencial en concreto. Un aspecto fundamental de esta solución es que toda la información sensible, es decir la que figura en los credenciales, se transmite en una comunicación punto a punto y además no se deja rastro de la misma en ninguna base de datos pública.

- **Hacer uso de la base de datos del estado**

  Tal y como comentamos en el apartado en el que estudiamos las características del proyecto *Hyperledger Fabric*, una de las ventajas clave era que nos proporcionaba una *CouchDB*. Se tratan de bases de datos que contienen los verificadores que mantienen el estado más reciente de la cadena de la red descentralizada. Actualmente cuando hacemos una búsqueda de un documento DID según su dirección en la cadena hacemos un recorrido lineal de la misma. Podemos ver como esta solución no es escalable conforme aumente de tamaño la cadena. Por ello la mejoría sería hacer uso de esta funcionalidad a la hora de resolver las transacciones del tipo *query*.

- **Redactar un documento que explique paso a paso el chaincode**

  Este punto de mejora se centra principalmente en el usuario que carece de conocimientos de informática. Volviendo a la generación del entorno de nuestro proyecto, la red estaba instalada de tal manera que todos los usuarios que quisieran participar en el canal común tenían que aceptar el *chaincode* propuesto. A partir de ese momento las únicas operaciones que se podían realizar sobre las cadenas eran las implementadas en dicho *chaincode*, esto quiere decir que si se quisieran evitar diversos tipos de fraudes relacionados con una utilización indebida de la red, el usuario debe leer el *chaincode* para así ser conocedor de exactamente qué operaciones se pueden realizar. En el caso de usuarios con bajo dominio de la informática esta tarea puede resultar complicada por lo que recomendamos redactar un documento que explique ese fragmento de código para así demostrar la transparencia de la implementación.

3.3. Opinión personal

A pesar de que la idea troncal que subyace a este trabajo ha permanecido constante, las identidades parciales, hemos tenido un largo recorrido de cambios en el enfoque que hemos seguido. Personalmente creo que para entender en profundidad la opinión que voy a exponer es necesario que haga un repaso cronológico con los todos los grandes hitos por los que hemos pasado.

Nos remontamos al mes de abril de 2019 en mitad del segundo cuatrimestre del curso 2018-2019, en ese momento hacía ya más de un año que tenía una sincera atracción por el tema de la ciberseguridad y la criptografía. Desde toda la información que buscaba por internet sobre los *exploits* más famosos de la actualidad hasta los relatos que me contaba mi padre en su trabajo como piloto en el ejército del aire en un departamento cercano al de ciberdefensa, sentí que la ciberseguridad era aquello a lo que me quería dedicar al menos en un futuro a medio plazo. Por ello tras reflexionar durante varias semanas, decidí mandar un correo el 18 de abril a Juan Carlos Fabero Jiménez pues me pareció un excelente profesor cuando me impartió clase de la asignatura "Redes", la cual está relacionada con temas que se tratan en ciberseguridad.

Todo esto desembocó en una reunión que tuvimos el lunes 29 de abril, dicha reunión fue mi primer contacto con el tema que iba a tratar pues la idea fue íntegra de Juan Carlos. La idea era clara: identidades parciales. Es decir buscar alguna forma de verificar credenciales por internet sin tener que proporcionar todos los datos asociados a una clave. El ejemplo que me hizo entender la problemática con la que estábamos tratando fue el de la FNMT (Fábrica Nacional de Moneda y Timbre), la cual actúa de CA de manera que permite asociar una clave pública con una serie de credenciales. La idea era permitir al usuario poseer una serie de credenciales propios y ser capaz de verificarlos para acceder a servicios de terceros a través de internet.

La marcha del trabajo no prosiguió hasta octubre de 2019, sin embargo, ya durante los meses de verano y el comienzo del primer cuatrimestre de 2019-2020, me sentía interesado por ver cómo podíamos solucionar el problema, por lo que hice una serie de apuntes estudiando conceptos de criptografía de clave pública como RSA/DSA, *hash* criptográficos como SHA-1, sistemas criptográficos como GPG y lo que pensábamos que era más importante, certificados X.509. Indico que pensábamos que dichos certificados eran la pieza más importante puesto que la idea inicial de la fragmentación de credenciales era la modificación de la especificación de los mismos para adaptar esta nueva funcionalidad. Comenzando ya de lleno en octubre empezamos a reflexionar sobre cómo se haría dicha modificación, como ayuda Juan Carlos me prestó el libro "Cryptography and Network Security, Principles and Practice". Sinceramente el libro me ayudó enormemente y lo recomendaría especialmente para alumnos que estén iniciándose en los temas de ciberseguridad, sin embargo, el libro me hizo entender una serie de dificultades graves que planteaban los certificados X.509. Al fin y al cabo se trata de una especificación de hace más de 20 años y no parecía lo suficientemente adaptable a lo que buscábamos pues soluciones del tipo tener una clave pública para cada credencial eran totalmente inviables.

Dos semanas más tarde el 29 de octubre de 2019 surge la gran revelación que tuve para la realización del trabajo. Tras días buscando información sobre el tema, me encontré con el proyecto *Rebooting Web of Trust*. Conforme fui informándome más y más sobre el proyecto, me di cuenta de que era exactamente el problema que queríamos resolver. En un principio me surgió un sentimiento de preocupación pues la idea del trabajo era avanzar en algo que fuese útil y que no estuviese resuelto, sin embargo este sentimiento rápidamente desapareció cuando conocí que *Rebooting Web of Trust* era un proyecto extremadamente novedoso (Diciembre de 2015 hasta la actualidad). Esto quería decir que primero de todo estábamos ante un tema que de gran interés a nivel de investigación y segundo todavía no existía ninguna implementación comercial de los conceptos del proyecto por lo que era el momento perfecto para hacer una aportación. La clave que hacía diferir la filosofía con la que habíamos seguido hasta el momento es que ahora no nos apoyaríamos en una modificación del X.509, sino que nos apoyaríamos en redes descentralizadas *blockchain*. Sobrecogido por un sentimiento de pasión sobre el trabajo, envié un correo ese mismo día a Juan Carlos para reunirnos el jueves de esa misma semana y explicarle la nueva propuesta.

Así comenzó la idea del trabajo tal y como se le da a conocer en este escrito al lector, pues la propuesta resultó ser un éxito. En ese momento era consciente de que el trabajo ya iba a ser más complicado de lo esperado porque en el grado de informática apenas se estudian conceptos de *blockchain*, pero mis ganas

de saber que habíamos encontrado un tema apasionante superaban cualquier temor de cómo seguir con el trabajo. El ritmo de trabajo constante comenzó con el segundo cuatrimestre en el cuál realizaríamos reuniones semanalmente en las que comentábamos todos los avances que había realizado durante la semana anterior. Empecé un largo recorrido de investigación de todas las herramientas que necesitábamos en el trabajo, aprendiendo cada vez más sobre temas de *blockchain*, estudiando el funcionamiento de los documentos DID que demostrarían ser fundamentales más tarde...

A mitad del desarrollo del trabajo llegué al punto que me resultó más dificultoso con diferencia: tenía que hacer la implementación de la red descentralizada que nos sirviese como base para resolver el problema de las identidades parciales. Lo importante era que en vez de hacer un prototipo de red descentralizada por mi parte, tal y como me sugirió Juan Carlos era mucho más interesante usar algún proyecto *open source* que ya implementase una y modificarlo. A raíz de esto encontré el proyecto *Hyperledger Fabric* y con ello la parte más complicada del trabajo.

La puesta en marcha de *Hyperledger Fabric* resultó se realmente difícil. Primero intenté hacer toda la instalación en un sistema operativo Windows pero después de un día entero probando sin éxito decidí instalar una máquina virtual e instalarlo sobre Ubuntu. Esto supuso un gran avance pues ya podía empezar a instalar las más de 4 herramientas distintas que requiere el proyecto, sin embargo, empecé a experimentar numerosos errores de ejecución pues dichas herramientas había que instalarlas con versiones específicas y no con las más modernas. Después de un fin de semana entero de pruebas sin parar fui al fin capaz de ejecutar el ejemplo "`Building my First Network`" que proporciona *Hyperledger Fabric*. Yo diría que a partir de aquí el trabajo fue a un ritmo ideal pues en este momento ya sabía exactamente qué tenía que hacer y además ya tenía la base para comenzar, por lo que lo que procedía ya no era más que una cuestión de trabajar las horas que hicieran falta para llevarlo a cabo.

Esto concluye un resumen de todas las fases por las que ha pasado este trabajo de fin de grado desde su génesis. Sinceramente creo que ha sido una experiencia enriquecedora académicamente en una gran variedad de aspectos. Por un lado he aprendido en profundidad sobre temas muy novedosos de la ingeniería informática como son las redes *blockchain*, por otro lado mi dominio de conceptos de ciberseguridad se ha visto ampliamente incrementado. La verdad es que nunca pensé en un principio que aprendería tanto de este trabajo y al final acabé con la impresión de que me ha enseñado mucho más que lo que estudio en una asignatura usual.

Me gustaría dedicar este último párrafo para agradecerle a Juan Carlos Fabero por su excelente labor como director. Desde un principio mostró un gran interés en el tema y esto provocaba que aumentase mi motivación por avanzarlo. Además no fue solo su gran interés, tal y como comenté, todas las semanas teníamos reuniones en las que hacíamos un seguimiento del contenido del TFG, esto me demostró que estaba realizando un extenso trabajo y que se preocupaba sinceramente por asegurar que la evolución del trabajo era la debería. Por todo ello, quiero agradecerle una vez más a Juan Carlos su labor, indicando que el nivel de la calidad en este trabajo no habría sido posible sin su aportación.

Capítulo 4

Apéndices

4.1. Infraestructura de Clave Pública - PKI

Las siglas PKI corresponde a *Public Key Infraestructure*, corresponde a un conjunto de políticas tanto software como hardware que se han de seguir para poder obtener una comunicación segura mediante el uso de certificados digitales.

Los protocolos de encriptación que se usan (principalmente RSA y DSA), recordamos que se basan en un sistema asimétrico de clave pública y clave privada. Esto provoca la aparición de un problema: tener alguna forma de verificar que una clave pública concreta, realmente pertenece a la organización/persona a la que queremos enviar el mensaje; ya que, un tercer individuo malicioso podría proporcionar una clave pública distinta suplantando una identidad. Para ello es necesario:

- CA: *Certificate Authority* son las organizaciones que emiten los certificados digitales, las más grandes que comprenden un total de 75 % de los certificados son Symantec, Comodo y GoDaddy.
- RA: *Registration Authority* su objetivo es el de promover el uso de un estándar que se ha escogido por convenio para facilitar las interacciones de la infraestructura.
- VA: *Verificacion Authority*, es decir una organización de confianza que se encargue de verificar que en efecto cada clave pública pertenece a la identidad correspondiente.

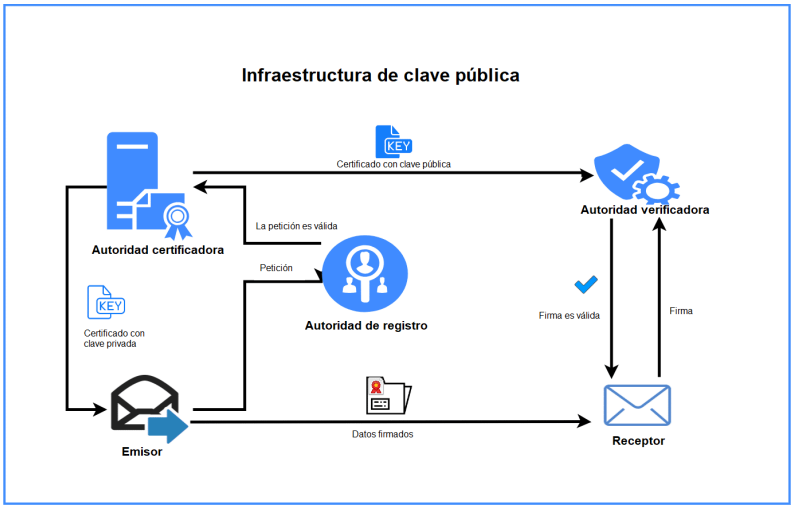


Figura 4.1: Esquema con los procedimientos definidos en la infraestructura de clave pública.

Bibliografía

[1] S. Appelcline, D. Crocker, R. Farmer, and J. Newton, *Rebranding the Web of Trust.* Rebooting the Web-Of-Trust, 2015.

[2] S. Appelcline, *Creating the new world of trust.* Rebooting the Web-Of-Trust, 2015.

[3] ConsenSys, *ConsenSys: Blockchain in Digital Identity.* Retrieved from `https://consensys.net/blockchain-use-cases/digital-identity/`, 2020.

[4] Hyperledger, *Prerequisites - Hyperledger-fabricdocs master documentation.* Retrieved from `https://hyperledger-fabric.readthedocs.io/en/release-2.0/prereqs.html`, 2020.

[5] Hyperledger, *Install samples, binaries and docker images - Hyperledger-fabricdocs master documentation.* Retrieved from `https://hyperledger-fabric.readthedocs.io/en/release-2.0/install.html`, 2020.

[6] H. Yang, *Cryptography Tutorials - Herong's Tutorial Examples.* WikiWikiWeb, 2002.

[7] D. Reed and L. Chasen, *Requirements for DIDs (Decentralized Identifiers).* Rebooting the Web-Of-Trust, 2016.

[8] D. Burnett, K. Ebert, A. Guy, D. Reed, and M. Sporny, *RWoT8 DID Specifcation Refinement.* Rebooting the Web-Of-Trust, 2019.

[9] H. Stahl, T. Capilnean, P. Snyder, and T. Yasaka, *Peer to Peer Degrees of Trust.* Rebooting the Web-Of-Trust, 2018.

[10] M. Sabadello, K. D. Hartog, C. Lundkvist, C. Franz, A. Elias, A. Hughes, J. Jordan, and D. Zagidulin, *Introduction to DID Auth.* Rebooting the Web-Of-Trust, 2018.

[11] D. Reed, L. Chasen, C. Allen, and R. Grant, *DID (Decentralized Identifier) Data Model and Generic Syntax 1.0 Implementer's Draft 01.* Rebooting the Web-Of-Trust, 2016.

[12] D. Reed, M. Sporny, M. Sabadello, D. Longley, and C. Allen, *Decentralized Identifiers (DIDs) v1.0, Core architecture, data model, and representations.* W3C, 2020.

[13] D. Gisolfi, M. Patel, and R. Radulovich, *Decentralized Identity Introduction.* IBM, 2020.

[14] Hyperledger, *Hyperledger - Wiki.* Retrieved from `https://wiki.hyperledger.org/`, 2020.

[15] E. Gerck, *Overview of Certification Systems: X.509, CA, PGP and SKIP.* Black Hat, 1998.

[16] Veres One, *Veres One - A Globally Interoperable Blockchain for Identity.* Retrieved from `https://veres.one/summary/`, 2020.

[17] Evernym, *Evernym - The Self-Sovereign Identity Company.* Retrieved from `https://www.evernym.com/about-evernym/`, 2020.

[18] Microsoft, *Microsoft - ixo Foundation.* Retrieved from `http://ixo.foundation/`, 2020.

[19] M. Sabadello, *A Universal Resolver for self-sovereign identifiers.* Medium, 2017.

[20] Hyperledger, *Using the Fabric test network.* Retrieved from `https://hyperledger-fabric.readthedocs.io/en/release-2.0/test_network.html`, 2020.

[21] Hyperledger, *Writing Your First Application.* Retrieved from `https://hyperledger-fabric.readthedocs.io/en/release-2.0/write_first_app.html`, 2020.

[22] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, *RFC 4880 - OpenPGP Message Format.* IETF, 1998.

[23] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, *RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* IETF, 2008.

[24] OpenPGP, *OpenPGP.* Retrieved from `https://www.openpgp.org/`, 2020.

[25] GnuPG, *The GNU Privacy Guard.* Retrieved from `https://gnupg.org/`, 2020.

[26] K. Rannenberg, D. Royer, and A. Deuker, *The Future of Identity in the Information Society.* Springer, 2009.

[27] S. Garfinkel and G. Spafford, *Web Security, Privacy & Commerce.* O'Reilly Media, 2002.

[28] S. Leible1, S. Schlager, M. Schubotz, and B. Gipp, *A Review on Blockchain Technology and Blockchain Projects Fostering Open Science.* Retrieved from `https://www.frontiersin.org/articles/10.3389/fbloc.2019.00016/full`, 2019.

[29] A. Rosic, *What is Blockchain Technology? A Step-by-Step Guide For Beginners.* blockgeeks, 2016.

[30] Deloitte, *What is a Blockchain?* Deloitte UK, 2016.

[31] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System.* metzdowd, 2008.

[32] PricewaterhouseCoopers, *Blockchain, a catalyst for new approaches in insurance.* PricewaterhouseCoopers, 2019.

Índice alfabético